DOCUMENT RESUME

ED 064 063 SE 013 436

AUTHOR	Brill, R. C.
TITLE	The Taxir Primer.
INSTITUTION	Colorado Univ., Boulder. Inst. of Arctic and Alpine Research.
SPONS AGENCY	National Science Foundation, Washington, D.C.
REPORT NO	Occas-Pap-1
PUB DATE	71
NOTE	79p.
EDRS PRICE	MF-$0.65 HC-$3.29
DESCRIPTORS	*Computers; *Information Retrieval; *Information Storage; *Information Systems; *Manuals; Resource Materials; Set Theory
IDENTIFIERS	Taxir

ABSTRACT

Taxir is an information storage and retrieval system designed for general use on electronic computing machines. The system is based on some simple notions in set theory which permit the data to be stored in a highly compressed form and rapidly retrieved by calculation rather than by the traditional comparison search. This technique results in significant savings in both machine storage space and machine execution time. This primer explains the Taxir system in detail from a user's point of view and assumes that the reader has no knowledge of mathematics or computers. (Author/CP)

ED 064063

U.S. DEPARTMENT OF HEALTH,
EDUCATION & WELFARE
OFFICE OF EDUCATION
THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL OFFICE OF EDUCATION POSITION OR POLICY

THE TAXIR PRIMER

R. C. BRILL

OCCASIONAL PAPER No. 1

INSTITUTE OF ARCTIC AND ALPINE RESEARCH ● UNIVERSITY OF COLORADO

SE 013 436

THE TAXIR PRIMER

by R. C. Brill

Occasional Paper No. 1
Institute of Arctic and Alpine Research
University of Colorado
Boulder

1971

Publication of this manual has been sponsored by the National Science Foundation (Atmospheric Sciences) under grant no. GA-15528 to Dr. R. G. Barry, Institute of Arctic and Alpine Research, University of Colorado.

TABLE OF CONTENTS

page

PREFACE. iv

About the Taxir System iv
About the Taxir Primer iv
Brief History of the Taxir System v

ACKNOWLEDGMENTS vi

1. INTRODUCTION TO TAXIR AND SET THEORY 1

Taxir Data Banks 3
Data Bank Design 5
Taxir Programs. 5
General Formats 7

2. BUILDING A TAXIR DATA BANK 8

STATEMENT TYPE: DEFINE DESCRIPTORS 8

ORDER Option. 9
NAME Option 11
FROM-TO Option 12
Equals Facility. 13

STATEMENT TYPE: DEFINE ITEMS 14

Fixed Field Format. 15
Free Field Format 16
SAME Format 17

STATEMENT TYPE: DEFINE AND PRINT ITEMS. 17

STATEMENT TYPE: Item Definition 17

STATEMENT TYPE: END OF ITEMS 18

A SAMPLE DATA BANK 18

3. QUERYING A TAXIR DATA BANK 21

STATEMENT TYPE: CONTROL VOCABULARY 21

BOOLEAN EXPRESSIONS 22

Boolean Operators 23
Taxir Boolean Operands (Type 1) 24
Taxir Boolean Operands (Type 2) 25
Taxir Boolean Operands (Type 3) 25
Taxir Boolean Operands (Type 4) 27
Combining Operators and Operands 27
The Use of Parentheses in Boolean Expressions. . . 28

STATEMENT TYPE: HOW MANY 32

STATEMENT TYPE: PRINT 33

Descriptor Lists 33
Line Combos 37
Print Fields. 40
Print Field Override Options (L and R Parameters) . 42
Print Field Override Options (F Parameter). . . . 43
SAME Option 44

HOW TO QUERY 45

STATEMENT TYPE: GENERATE 49

File Structure 50
Other Possible Applications. 51

4. CORRECTING A TAXIR DATA BANK. 52

STATEMENT TYPE: CORRECTION. 52

STATEMENT TYPE: DELETE STATE 53

STATEMENT TYPE: DELETE ITEMS 53

STATEMENT TYPE: DEFINE MORE DESCRIPTORS 54

5. MISCELLANEOUS TAXIR STATEMENTS 55

STATEMENT TYPE: ID 55

STATEMENT TYPE: END 55

STATEMENT TYPE: TIME. 56

STATEMENT TYPE: MEMO. 56

STATEMENT TYPES: READ DATA BANK and WRITE DATA BANK . . 57

APPENDIX A: TAXIR ERROR MESSAGES 59

Taxir Error List 60

APPENDIX B: PRELIMINARY PROBLEMS 64

Data Bank Design 64
Fitting the Taxir System to the Data Bank. 65
Running Taxir under the Operating System 66
Taxir System Conversion. 67
Summary 68

INDEX 69

PREFACE

About the Taxir System

Taxir is an information storage and retrieval system designed for general use on electronic computing machines. The semantic content of the information fed to it has no bearing on its operation. If the information can be structured in the manner described in Chapter 1, then the Taxir system is capable of handling it, and most of the information that is gathered in all fields of human endeavor can be so structured.

The system is based on some simple notions in set theory which permit the data to be stored in a highly compressed form and rapidly retrieved by calculation rather than by the traditional comparison search. This technique results in significant savings in both machine storage space and machine execution time.

The Taxir user addresses the system in a high level language, somewhat resembling English, which permits him complete control over building, updating, and querying data banks. The querying section of the language, which enables the user to retrieve desired portions of his data, is the language of boolean algebra, adapted so that terms of the user's own choosing may serve as operands in boolean expressions of any degree of complexity.

The responses to queries are ordered alphabetically, numerically, or on any other ordering criterion defined by the user, and arranged in a convenient hierarchical structure of the user's choosing.

About the Taxir Primer

(This primer explains the Taxir system in detail from a user's point of view and assumes that the reader has no knowledge of mathematics or computers.) Readers who are well versed in these topics will, I hope, bear with the simple discussions of sets and boolean algebra which my readers who are not so well versed will find absolutely necessary for an understanding of the system. There are many persons who are not scientifically trained who nevertheless find themselves in charge of large masses of information and in great need of tools for controlling them. Taxir is such a tool and it is my belief that the use of this tool can be learned by any intelligent motivated person with or without a scientific education who is willing to read and study this primer.

Brief History of the Taxir System

The system owes its origin to the long-standing interest of plant taxonomist David J. Rogers in computer aided classification and museum curation. During the early 1960's Dr. Rogers assembled a research team to study the classification process. Several computer programs were developed over the period 1963 to 1966, most notably the similarity-clustering program (based on graph theory and set theory) and the character analysis program (based on information theory and set theory), both largely contributions of mathematician George Estabrook. Although the original impetus was to classify biological organisms, Estabrook's approach was to regard classification as a general procedure for grouping similar objects into clusters, whatever these objects may be and whatever characteristics of the objects may be chosen as the basis for similarity and difference. Both of these programs are still much used by taxonomists, but they are increasingly being used by other workers outside systematic biology, such as geologists, psychologists, ecologists, etc.

In 1967 Dr. Rogers and his team were awarded a grant from the National Science Foundation to develop a computerized information storage and retrieval system for systematic biology, principally to facilitate the curation of botanical and zoological collections. Again, the team took a generalized approach and extending the concepts developed during the earlier work in classification, Brill and Estabrook designed and programmed (for the CDC 6400) the Taxir system. Taxir stands for Taxonomic Information Retrieval, a name which reflects the original intentions of the project. These intentions were fully realized and a number of Taxir museum applications are now in existence. But a great deal more was achieved in the process and thanks to Taxir's generality we are now able to offer the system to all who wish to retrieve at will select portions from a large mass of data, whatever the nature of that data. As with the classification programs, workers outside systematic biology have begun to use the system and it is our hope that not only scientists, but businessmen and administrators, will recognize the common structure underlying their information problems and give the Taxir system a wide acceptance.

R. C. Brill
Boulder, Colorado
April, 1971

ACKNOWLEDGMENTS

Above all to David J. Rogers and the staff of the Taximetrics Laboratory at the University of Colorado for the support and encouragement given to Estabrook and myself during the original research, design, implementation, and testing of the Taxir system.

To Roger G. Barry of the Institute of Arctic and Alpine Research at the University of Colorado for support and encouragement during the design, implementation, and testing of improvements to the Taxir system and for the publication of the Taxir Primer.

To the National Science Foundation, who in the form of grant no. GN-656 (David J. Rogers, Principal Investigator) and grant no. GA-15528 (Roger G. Barry, Principal Investigator) sponsored the developments referred to above.

To George Estabrook, George W. Nace, and Roger G. Barry for suggestions about and corrections to the Taxir Primer.

To George W. Nace, Department of Zoology, The University of Michigan, and his staff for typing and proofreading of the Taxir Primer under the support of National Science Foundation grant no. GB 8187.

To William Walden, Director of the Computing Center, Washington State University, and his staff for undertaking the conversion of the Taxir system to the IBM 360/65.

To the following Taxir users: S.G. Appan, R.G. Barry, L. Blick, J.M. Clark, S. Dietz, H. Fleming, N. Hairston, M. Hale, L.W. Hudson, W. Klein, A. Kluge, A. Löve, T. Mason, F. Richardson, T. Stewart, W. Weber, R. Westdyke, and G.F. White, for using the Taxir system while it was still under development and without benefit of a user's manual. The feedback from these experiences has given rise to many of the improvements now present in the system.

---o---

1. INTRODUCTION TO TAXIR AND SET THEORY

It is a strange property of the human mind to impose upon the flux of its sensory input an exhaustive taxonomy which structures the world for us into things. Each thing has its name and its attributes. Pencil. Paper. Blue pencil. Red pencil. Writing paper. Wrapping paper. Thus we are shielded from reality by an elaborate metaphor and all that we know of the world is cloaked in its terms. To behave as though the illusion were the reality, as most of us do most of the time, is to be sane.

For those of us who work with masses of data, set theory provides a simple, powerful and practical tool for manipulating information about things. A set is a collection of things. The people in a room, the books in a library, the fish in a lake, the specimens in a museum are all sets. The things of which sets are composed are called elements or members. A pair of dice is a set with two members. A single rose is a set with one member. All the money in an empty pocket is a set with no members. The universal set (written as U) can be defined as the set of all the elements under consideration. If U = a certain wolf pack, then we can find subsets of U, such as the female wolves or the young wolves or the females and young taken together or those wolves who are just now facing east, etc., etc. Some of these subsets may be equivalent to (that is, have the same elements as) the universal set. For example, if U = the wolf pack, then the subset with fur = U. Some of these subsets may be equivalent to the null or empty set (written as Ø), a set with no elements. If U = the wolf pack, then the subset with bankbooks = Ø. Sets are themselves things and so there are sets whose members are sets, like the set of all wolf packs which are in turn sets of wolves.

This simple, intuitively grasped notion of things structured into sets and sets themselves hierarchically structured into other sets lies at the foundation of all our mathematics. The recent realization that this is so has led to the introductory mathematics now taught to our youngest school children. The core of this approach is to teach before all else the simple properties of sets and the concept of number. In the mid-nineteenth century George Boole published the first statement of the algebra of logic. The concepts of Boole have been extended by other workers and in various contexts they are known by the names boolean algebra, the algebra of sets, propositional calculus, logical arithmetic and by other names as well. In our own context of data manipulation, we can consider boolean algebra as a technique for combining and altering sets to form other sets. Using this algebra to operate on sets is reminiscent of the way we use ordinary arithmetic to operate on numbers, only easier.

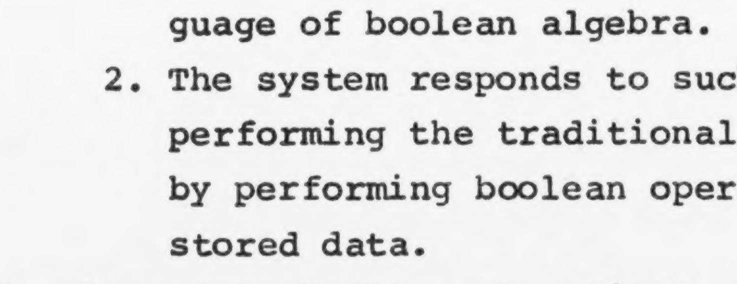
Our computing machines all perform ordinary arithmetic on numbers. In the machine these operations are actually composed of a series of simpler boolean operations on the bits which represent the numbers. Computing machines are at the circuitry level boolean algebra machines *in toto*. And at the programming level the machine user may perform boolean operations on variables of his own choice. It is this fundamental property of computing machines which makes the Taxir system possible and which the designers of Taxir have exploited in 2 ways:

1. The language which the Taxir user employs to select desired subsets of his data for printout is the language of boolean algebra.
2. The system responds to such user requests, not by performing the traditional linear file search, but by performing boolean operations directly on the stored data.

As these same boolean operations are the simplest and fastest in the machine, this technique accounts for Taxir's exceptional retrieval speed. See Estabrook, G.F. and R.C. Brill, 1969, *The Theory of the Taxir Accessioner*, Mathematical Biosciences, *5*:327-340.

The elements of a set U can be represented pictorially as all the points bounded by a rectangle (or any other closed plane figure). The elements of the various subsets of U can be represented as the points bounded by circles (or other closed plane figures).

U = all the people in Room 7



Some of the subsets may be:
A = those with hats
B = those with coats
C = those named Robert
D = those over 20
E = those over 40

No one named Robert has a hat. This reflects the fact that sets A and C are disjoint, i.e., have no elements in common. Other disjoint pairs of sets are (A,D), (A,E), (B,C), (B,D), (B,E), (C,D), and (C,E). Some people have hats, some have coats and some have both, reflecting the fact that sets A and B overlap, i.e., some but not all of the elements of A are also elements of B, and vice versa. All those over 40 are also over 20, which reflects the fact that set D contains set E, or to phrase it another way, E is a subset of D. (All elements of E are also elements of D.) E is also a subset of U and so are A, B, C and D.

There are 3 ways of establishing the membership of a set P.

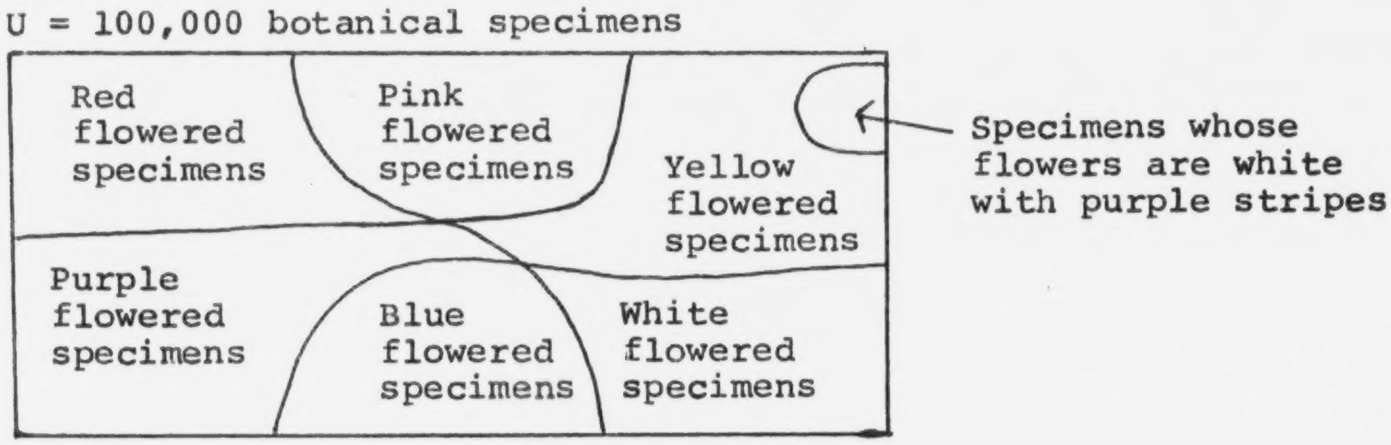
1. by naming the elements of the set, such as, P is the set whose members are Peter, Paul and Mary.
2. by naming some condition or property or rule that selects the elements of U that qualify for membership in the set P, such as, P is the set of those people with purple neckties.
3. by performing one or more of the 3 basic operations of boolean algebra on already existing sets, such as, P is the set of those people with purple neckties and incomes over $10,000/year. Q is the set whose members are Peter or Paul or not those with purple ties.

These operations are defined and illustrated in Chapter 3.

Taxir Data Banks

A data bank is a collection of things, i.e., a set. Each element of a data bank corresponds to an element in some other set of interest such as a collection of museum specimens, a group of ecological habitats, the contents of a warehouse, the employees of a company, the books of a library, the students of a university, the results of an experiment, the findings of an oceanographic survey, the inventory of a store, etc., etc. Each element of a data bank is called in Taxir terminology an item and is composed of all the pieces of pertinent information describing the corresponding element in the set of interest. Although there is a significant conceptual difference between an object, such as a museum specimen, and the recorded data in the bank concerning this object, it is simple and convenient in daily usage to let the word item designate either one, the object itself or its information abstract.

The pieces of information which comprise an item belong to information categories called descriptors. A descriptor may be thought of as a criterion for dividing all the items in a data bank into mutually exclusive (disjoint) subsets called descriptor-states or sometimes just states. For example, if the items in a data bank are botanical specimens, then with respect to the descriptor FLOWER COLOR each specimen will be assigned to a subset (state) in accordance with the color of its flower. All the items with red flowers will constitute the membership of a subset called FLOWER COLOR, RED. All the items with pink flowers will constitute the membership of a subset called FLOWER COLOR, PINK. Etc. If a specimen is discovered to have white flowers with purple stripes, it is not assigned to both the states WHITE and PURPLE, but rather, a new state which might be called WHITE WITH PURPLE STRIPES is admitted to the list of recognized states. In this way the states of a descriptor are guaranteed to be mutually exclusive.


U = 100,000 botanical specimens
Red flowered specimens
Pink flowered specimens
Yellow flowered specimens
Specimens whose flowers are white with purple stripes
Purple flowered specimens
Blue flowered specimens
White flowered specimens

Any data entered into the Taxir system will be cast into the set theoretic data structure described above, but the information content of the items, descriptors and descriptor-states, as well as their quantity and their identifying names, are strictly under the control of the Taxir user. Virtually all information can be rather naturally and easily structured as sets, but this set theoretic data organization provides more than a convenient means for storing information. As will presently be shown, it provides us with a powerful means of selective retrieval.

Data Bank Design

Before the potential Taxir user can put the Taxir system to work, he must design a data bank or a set of data banks. In his role as data bank designer he must ask and answer the following essential questions:

1. How many data banks will cover my needs?
2. In each of these data banks, what conceptually is an item?
3. How many items can I expect in each data bank?
4. What pieces of information should be preserved about these items? The answer to this will take the form of a list of descriptors for each data bank.
5. In each descriptor chosen, what will be the range of expression? The answer to this will take the form of either a list of states or an estimate of the number of states for each descriptor chosen.
6. What kind of queries can be expected? That is, what do I want to do with this mass of information? In what ways do I want to use the power of selective retrieval?

The main purpose of this primer is to instruct the user who has already solved his design problem and who is now ready to put the Taxir system to work. For the user who still has to answer the above questions, this primer is an essential guide, for it illuminates the possibilities raised by these questions, but it is not by itself enough. Appendix B examines the data design problem and some related problems in more detail and sets forth some suggestions.

Taxir Programs

The Taxir user communicates with the Taxir system by writing a Taxir program, which consists of a series of statements in the Taxir language. These statements are prepared for machine execution by punching them on cards (or typing them at an input terminal) and feeding them to the Taxir system. This primer will use the term card to mean either a punch card or a line typed on a terminal. The system reads and executes each statement before treating the next statement in the Taxir program. There are no facilities for conditional jumps, hence no program branching or looping. The program is executed sequentially from the first to the last statement at which time program execution is finished.

There are 19 statement types in the language, each of which is associated with a statement name. This primer describes in detail each statement type in the Taxir language and how to use it, as well as discussing how to combine statements into Taxir programs.

Some general rules about statements are:

1. Each statement begins on a new card.
2. Each statement (except for the type called Item Definition) begins with its statement name. In some cases, this is the entire statement. In other cases, the statement name is followed by additional information.
3. A statement name must be complete on the first card of the statement which it introduces.
4. All blanks embedded in statement names (such as DEFINE DESCRIPTORS), descriptor names (such as GEOGRAPHICAL LOCATION), or descriptor-state names (such as NOVA SCOTIA) are considered by the system to be a part of these names and must be included each time such names appear in Taxir statements.
5. Leading and trailing blanks around names or punctuation marks are optional. In a few special cases leading and trailing blanks are required and these are carefully stipulated in the text.
6. With regard to the number of cards permitted per statement, the Taxir language falls into 3 classes.

   A Class I statement may extend over as many cards as are desired and must be terminated by an asterisk.

   A Class II statement must be complete on one card and need not be terminated by an asterisk.

   Class III covers only Item Definitions under the FIXED option. The number of cards in such a statement is determined by the field definitions of the governing DEFINE ITEMS statement. No terminating asterisk is needed. This is fully explained in the section called STATEMENT TYPE: DEFINE ITEMS.
7. Class I and Class III statements are treated as though the cards composing them were laid end to end to form one long card. For example, card 1/col. 80 is considered to be contiguous with card 2/col. 1.

Class I Statement Names

CONTROL VOCABULARY
CORRECTION
DEFINE AND PRINT ITEMS
DEFINE DESCRIPTORS
DEFINE ITEMS
DEFINE MORE DESCRIPTORS
DELETE ITEMS
DELETE STATE
GENERATE
HOW MANY
Item Definition (under FREE option)
MEMO
PRINT

Class II Statement Names

END
END OF ITEMS
ID
READ DATA BANK
TIME
WRITE DATA BANK

Class III Statement Names

Item Definition (under FIXED option)

General Formats

Associated with each statement type in the text is a general format, which might be described as an expression which attempts to summarize the full range of the statement's powers. Throughout the primer, wherever these general formats appear, the following conventions are employed:

1. Brackets { } enclose choices. When composing a statement, the user may choose one of the alternatives listed vertically within the brackets.
2. Symbols in capital letters and punctuation marks (except ...) must appear in a real statement exactly as they appear in the format.
3. Symbols in lower case letters represent a set of choices and these choices are always defined in the text.
4. A lower case b associated with a word in capital letters, such as bWITHb, designates a required blank.
5. An ellipsis (...) designates a series of indefinite length, such as d(p), d(p),...d(p). The space occupied by the ellipsis may be filled in with additional instances of d(p),.

---o---

2. BUILDING A TAXIR DATA BANK

Assuming now that the user has made all the necessary preliminary design decisions which set the structure of his data bank, he may now bring his data bank into existence by a Taxir program which includes a sequence of the following statement types:

```
DEFINE DESCRIPTORS
DEFINE ITEMS
Item Definition
      ↓
Item Definition
END OF ITEMS
```

The DEFINE DESCRIPTORS statement is basically a list of the descriptors chosen by the user together with some parameters that teach the Taxir system how to interpret the data that follow. The DEFINE ITEMS statement tells the system that there is a block of Item Definitions awaiting definition on either cards, tape or disk, and it teaches the system the physical layout of data as they appear on the card, tape or disk records. There follows an Item Definition for each item being entered into the bank at this time. An Item Definition is basically a list of all the states to which a single item has been assigned (one state for each descriptor in the bank). The Item Definition series is terminated by an END OF ITEMS statement, which merely serves to inform Taxir that the end of the series has been reached.

STATEMENT TYPE: DEFINE DESCRIPTORS

This statement appears once and only once in the life of a data bank and its appearance marks the birth of the data bank. This statement imposes structure on the stream of input data that may follow at intervals throughout the life of the bank.

Its general format is:

```
DEFINE DESCRIPTORS d(p), d(p),...d(p)*
```

d = descriptor name. The user may choose as a descriptor name any sequence of from 1 to 90 characters selected from the available character set, but not containing any of the following characters:

( ) * ,

and not containing any of the following character strings:

bANDb
bORb
bNOTb
bFROMb
bTOb
bRESULTb
bSAMEb
bFORb

A descriptor name must contain at least one non-blank character. Leading and trailing blanks are ignored and not considered to be part of the name. Blanks embedded in the name itself are considered to be part of the name. Each descriptor in a data bank must bear a unique name.

Examples: EMPLOYEE NAME
DEPARTMENT
YEAR OF PURCHASE
1ST ESTIMATE
2ND ESTIMATE
A+B
X
..$--/8..
ABCDEFGHIJKLMNOPQRSTUVWXYZ

p = parameter list. This may be broken down into a more detailed general format, as follows:

$$\boxed{\left\{\begin{array}{l}\text{ORDER, ds, ds,...ds}\\ \text{NAME, est}\\ \text{FROM a TO b }\underbrace{\text{BY c,}}_{\text{optional}}\ \underbrace{\text{IN label,}}_{\text{optional}}\\ = \left\{\begin{array}{l} \text{i}\\ \text{j}\end{array}\right\}\end{array}\right\}}$$

Each descriptor will be defined under one of three options: ORDER, NAME or FROM-TO.

ORDER Option

$$\boxed{\text{d(ORDER, ds, ds,...ds)}}$$

ds = descriptor-state name. These names conform to the same rules as descriptor names except that the character strings bRESULTb, bSAMEb, and bFORb are permitted. Following the work ORDER is the complete list of states for the descriptor d, separated by commas. The first (left-most) name in the list is assigned by Taxir the code number 1, the second name in the list is assigned the code number 2, etc. In the Item Definitions, in the field reserved for the states of this descriptor, Taxir will expect to find either one of the ds names from the list above or one of the code numbers assigned to these names. There need be no consistency in the practice. In one Item Definition the name may appear and in another the code may be used.

Example: In a data bank with the descriptor MONTH (ORDER, JANUARY, FEBRUARY, MARCH, APRIL, MAY, JUNE, JULY, AUGUST, SEPTEMBER, OCTOBER, NOVEMBER, DECEMBER), the numbers 1 through 12 may be used to stand for the months of the year when defining items. In this example, the keypunch operator will find it easier to punch numbers than the full names of the months, especially as this particular correspondence between names and numbers is already in common use.

Example: GRADE (ORDER, A, B, C, D, E, F). In this case the names A through F are as easy to punch as the codes 1 through 6, easier in fact, as the keypuncher need not learn a name-to-code correspondence, but can punch the grades as they appear on the source documents.

The prime consideration in deciding whether to punch names or codes should be the keypuncher's convenience. By far the largest bulk of Taxir statements in any sizable data bank will be Item Definitions, i.e., the entering of new data into the bank. Anything that can be done to ease the strain on the keypunch operator will increase the speed and the accuracy of the punching.

The principal advantage of the ORDER option is that it establishes an order to the states, so that later when querying, the user may take advantage of the FROM ds1 TO ds2 facility.

Examples: Assuming a data bank containing the 2 ordered descriptors of the 2 previous examples, we might exploit their order in the following queries:

```
HOW MANY ITEMS ARE THERE WITH MONTH, FROM JUNE TO SEPTEMBER*
HOW MANY STUDENTS HAVE GRADE, FROM A TO C*
```

Taxir will understand the FROM-TO range in the first instance to include the months of June, July, August and September, and in the second instance to include the grades A, B and C. There is a full discussion of the FROM dsl TO ds2 querying facility in the section on BOOLEAN EXPRESSIONS, Taxir Boolean Operands (Type 3).

There exists the unlikely possibility that a ds name will be the same as one of the code numbers. Example: POWER (ORDER, 1, 2, 4, 8, 16, 32, 64). If Taxir picks up a 4 in the appropriate field of an Item Definition, this could refer to the name 4 (code = 3) or to the code 4 (name = 8). Taxir resolves ambiguities of this type by treating the number in question as a name and not as a code number.

NAME Option

d(NAME, est)

For the descriptor defined under the NAME option there is no preset ordered list of permissible descriptor-state names. Rather whatever names are found in the Item Definitions, in the field reserved for the states of this descriptor, shall be accepted as valid state names, provided they conform to the rules for descriptor-state names already mentioned. Codes may not be used in place of names when defining items and the FROM dsl TO ds2 facility may not be used when querying.

This option is useful for cases where a full list of all possible state names cannot be anticipated at the time of the data bank creation.

Example: In a data bank where the items are students of a university, the descriptor NAME OF STUDENT would almost necessarily be defined under the NAME option. The name of a student will be punched in the appropriate field of each Item Definition.

The word NAME must be followed by the parameter est, which is an estimate in the form of a positive integer of the maximum number of states expected in this descriptor. This is necessary because storage space must be reserved in the machine for a dictionary of these expected state names. The upper limit on this estimate will be determined by available machine space.

Examples: NAME OF STUDENT (NAME, 5000)
GEOGRAPHICAL LOCATION (NAME, 50)
COLOR OF SPECIMEN (NAME, 12)
DEPARTMENT (NAME, 80)

FROM-TO Option

$$\boxed{\texttt{d(FROM a TO b }\underbrace{\texttt{BY c}}_{\texttt{optional}}\ \underbrace{\texttt{IN label}}_{\texttt{optional}}\texttt{)}}$$

This option is ideal for the common case in which the states of a descriptor are strictly numeric and can be ordered as a series whose successively larger members differ from their neighbors by the same amount. To define such a set of numbers 3 values are needed. These are *a* (the numerically smallest value of the series), *b* (the largest value), and *c* (the increment by which the series increases stepwise from *a* to *b*).

Note that it is possible to name 3 values such that *b* is not a member of the set defined. FROM 0 TO 99 BY 2 is such a case. Taxir will interpret the set to be 0, 2, 4, 6,...98, terminating the set with the largest value obtained (by repeatedly adding the increment *c*) that does not exceed *b*.

a and *b* may be negative, 0 or positive. If negative, use a minus sign. If 0 or positive, use no sign. They may be integers, fractions or mixed numbers and must be expressed in decimal notation, although integers need not have a decimal point.

c follows the same rules except that it must be positive. *c* may be omitted, in which case Taxir understands the increment to be 1.

When printing the states of a descriptor defined under this option, Taxir prints as many digits to the right of the decimal point as the *a*, *b* or *c* parameter with the most such digits.

label is a name which is subject to the same rules as descriptor-state names, except that it may not exceed 10 characters in length. When printing the states of a descriptor defined with a label, Taxir also prints the label following the value. The purpose of the label is to supply a unit of measurement such as LBS., FT., MPH, etc.

Examples: YEAR(FROM 1900 TO 1980)
TEMPERATURE (FROM -50 TO 120 IN DEGREES F.)
WEIGHT (FROM 0 TO 100 BY 5 IN LBS.)
COST (FROM 2.50 TO 10000 BY .01 IN $)
PROBABILITY (FROM 0 TO 1 BY .001)
ITEM NO. (FROM 1 TO 100000)

Equals Facility

$$\boxed{d(= \left\{\begin{matrix} i \\ j \end{matrix}\right\})}$$

Whenever two or more descriptors in the same data bank have the same states and differ only in their descriptor names, the equals facility may be used. The user follows the = sign with either $\underline{i}$, the descriptor number, or $\underline{j}$, the descriptor name, of any descriptor appearing earlier in the same DEFINE DESCRIPTORS statement and to whose states the user wishes to equate the states of the present descriptor. The first (left-most) descriptor name in the DEFINE DESCRIPTORS statement is descriptor number 1, the second name is descriptor 2, etc.

Example: DEFINE DESCRIPTORS MONTH OF PURCHASE (ORDER, JAN., FEB., MAR., APR., MAY, JUNE, JULY, AUG., SEPT., OCT., NOV., DEC.), MONTH OF SALE (=MONTH OF PURCHASE), DAY OF PURCHASE (FROM 1 TO 31), DAY OF SALE (=3), DAY OF JUDGMENT (=3)* This last descriptor can equally well be defined as DAY OF JUDGMENT (=4) or DAY OF JUDGMENT (=DAY OF PURCHASE) or DAY OF JUDGMENT (=DAY OF SALE).

The equals facility is not to be construed as a separate option. The option will be that of the $\underline{i^{th}}$ descriptor. This facility not only provides the user with a shorthand form of expression, but more importantly it permits Taxir to establish a single dictionary of state names for any equated descriptors. In the case of descriptors defined under the NAME option with a large estimated number of states, this will save considerable machine storage space. It is very much to the user's advantage to use this facility whenever the opportunity presents itself.

There exists the unlikely possibility that a descriptor name will be the same as one of the descriptor numbers. Example: DEFINE DESCRIPTORS 2(NAME, 100), J(FROM 1 TO 10), K(=2)* The =2 could refer to the descriptor whose name is 2 (and whose number is 1) or to the descriptor whose number is 2 (and whose name is J). Taxir resolves such ambiguities by treating the number in question as a descriptor name.

STATEMENT TYPE: DEFINE ITEMS

```
                     ⎧CARDS⎫        ⎧bFIXED c, c,... c⎫
DEFINE ITEMS FROM    ⎨TAPE ⎬ noise  ⎨bFREE  f, f,... f⎬*
                     ⎩DISK ⎭        ⎩bSAME            ⎭
```

This statement informs Taxir that it can expect to find a series of Item Definitions. If the option CARDS is chosen, the Item Definition series with its terminating END OF ITEMS statement should be on cards, following immediately in the deck after the DEFINE ITEMS statement. If the option TAPE (or DISK) is chosen, the Item Definition series should exist as a file on a magnetic tape (or disk). This file should be the tape (or disk) analogue of a card deck, that is, written in alphanumeric (or as it is sometimes called, BCD) mode and set up in 80 column records. It should be terminated either by the END OF ITEMS statement or by an end-of-file mark or both. This tape (disk) should be already positioned to read the first record (card image) of the file. This can be accomplished through the control language of the operating system at the user's computer installation and should be done earlier in the run before the Taxir system takes control of the machine.

The term noise means that the user may insert here any combination of characters except the words bFIXED, bFREE or bSAME. Noise is ignored by Taxir. Its purpose is to improve the readability of the statement (not for the machine, but for humans). It is always permissible to omit the noise, but the noise terminator (bFIXED, bFREE or bSAME) must appear.

The options in the second set of brackets describe to Taxir the physical layout of the data on the expected card, tape or disk records. Inasmuch as a tape or disk record is a card image, data formats for all three media will be the same and can be discussed in terms of cards and card columns. We can also assume that these cards are conceptually laid end to end so that card 1/col. 80 is contiguous with card 2/col. 1. In like manner each card of an Item Definition, and there may be any number of these, is treated as continuous with its neighboring cards.

The Item Definition is divided into fields. There should be a field reserved for each descriptor in the bank. Each field should contain one of the states of the descriptor for which the field is reserved.

Fixed Field Format

If the option FIXED is chosen, then a field is defined as being the set of contiguous card columns beginning at card x/col. y and ending at card x'/col. y', where the latter comes after (to the right of) the former in the card continuum.

The parameter c defines a field. If the field is but a single card column, c may take the form card x/col. y (example: 2/8). If the field is larger than a single column, c may take the form card x/col. y - card x'/col. y' (example: 2/8-2/20).

c may also take the following forms:

	Examples:
col. y	19
col. y - col. y'	45-49
col. y - card x'/col. y'	7-2/25

In these cases, which are characterized by the omission of one or both of the card numbers, Taxir will assume that the missing card number is 1.

The first (left-most) c parameter defines the field for descriptor 1 (the first, or left-most, descriptor in the DEFINE DESCRIPTORS statement), the second c parameter defines the field for descriptor 2, etc. There must be a c parameter for each descriptor in the bank. The order of the c parameters creates the necessary link between the data in the Item Definitions and the descriptors, thus reserving a field in the Item Definition for each descriptor. Any portions of the Item Definition cards not specified in a c parameter will be ignored.

The fields should be designed to accomodate the largest expected state name or code. The minimum field length is 1 card column and there is no maximum. A practical upper limit is 90 card columns as Taxir will not accept a name longer than 90 characters. However, a field may always be larger than its contents. Remember that leading and trailing blanks are ignored, so a state name or code may sit anywhere inside its field, not necessarily beginning in the first column of the field.

These field definitions may be chosen to suit the user's convenience, but once chosen they are fixed for the current batch of Item Definitions. The next time the user wishes to add a batch of items to the bank he is free to choose an entirely different data format. In this way data decks or tapes originally designed for other uses may, if they contain sufficient valid data, be accepted by Taxir without conversion to some "standard" input format.

Under the Fixed Field Format option a field is always the same length and always appears in the same position in each Item Definition. This permits the keypunch operator to set up a drum program card for uniform punching.

Free Field Format

It is sometimes convenient to have a more flexible arrangement than the Fixed Field Format described above. Under the Free Field Format option the fields are maintained in the same order from item to item, but may vary in length and position. This is done simply by separating the fields by commas and terminating the Item Definition with an asterisk.

If the option FREE is chosen, then a field is defined as being the set of contiguous card columns between successive commas. (Or between card 1/col. 1 and the first comma. Or between the last comma and the terminating asterisk.)

The parameter f defines a field by its order of appearance in the Item Definition. f may take the form field order no. (example: 3, which indicates the third field, counting from the left in the card continuum) or field order no. - field order no. (example: 2-4, which indicates the three fields 2, 3 and 4).

The first (left-most) field order no. defines the field for descriptor 1, the second field order no. defines the field for descriptor 2, etc. There must be a field order no. for each descriptor in the bank. Any fields not specified in an f parameter will be ignored.

SAME Format

This option simply tells Taxir to expect the same data format that was specified in the last DEFINE ITEMS statement. It makes no difference if the last DEFINE ITEMS statement appeared earlier in the same run or in some previous run. Clearly this option cannot be used if this is the first batch of items to be defined into the data bank.

STATEMENT TYPE: DEFINE AND PRINT ITEMS

```
DEFINE AND PRINT ITEMS FROM [same parameters as DEFINE ITEMS]*
```

This is a minor variation of the regular DEFINE ITEMS statement. In addition to performing the tasks described above, this variant will generate a complete printout of the Item Definition series. This increases the length of computer time expended in creating the data bank and therefore increases its cost. On the other hand it is often useful to have a copy of the raw data, particularly for tracking down errors.

STATEMENT TYPE: Item Definition

Under the FIXED option the general format for the Item Definition statement is as follows:

```
|ds|    |ds|...|ds|
```

The vertical lines are meant to indicate that each ds appears in the fixed field defined in the governing DEFINE ITEMS statement.

Under the FREE option the general format is:

```
ds, ds,...ds*
```

If the descriptor to which ds belongs was defined under the ORDER option, then ds must be either one of the descriptor-state names defined in the DEFINE DESCRIPTORS statement or the code number for that state.

If the descriptor to which ds belongs was defined under the NAME option, then ds must be a descriptor-state name conforming to the rules already stated for such names.

If the descriptor to which ds belongs was defined under the FROM-TO option, then ds must be one of the numbers in the set defined. The only non-numeric characters permitted are the minus sign and the decimal point.

In addition to the descriptor-states defined by the user, each descriptor is automatically assigned by Taxir a state called UNKNOWN. Whenever information is missing leave the appropriate fields blank and Taxir will assign the item in question to the UNKNOWN state of the descriptors involved. Under the FREE option no blanks are actually necessary. The 2 field defining commas may be in adjacent columns.

STATEMENT TYPE: END OF ITEMS

```
END OF ITEMS
```

If the Item Definition series is on cards, this statement must follow the series in order to inform Taxir that the series is finished. If the Item Definition series is on tape or disk the series must end with either this statement or an end-of-file mark or both.

A SAMPLE DATA BANK

The XYZ Company is using Taxir to keep track of its money transactions. The managers of a real company would probably monitor their business in greater detail than this, but examples seem to illustrate their points more clearly if they are simplified. In this spirit let us assume that XYZ operates on a small scale, preserving only the following information categories. DEFINE DESCRIPTORS TRANSACTION NO. (FROM 1 TO 50000), TRANSACTING PARTY

```
(NAME, 2000), ADDRESS OF TRANSACTING PARTY (NAME, 2000), NATURE
OF TRANSACTION (ORDER, EXPENSE, INCOME), YEAR OF TRANSACTION (FROM
1950 TO 1990), MONTH OF TRANSACTION (ORDER, JAN., FEB., MAR., APR.,
MAY, JUNE, JULY, AUG., SEPT., OCT., NOV., DEC.), DAY OF TRANSACTION
(FROM 1 TO 31), DESCRIPTION OF GOODS (NAME, 5000), DELIVERY STATUS
(ORDER, DELIVERED, PARTLY DELIVERED, UNDELIVERED), TOTAL AMOUNT
(FROM 0 TO 10000 BY .01 IN $), PAYMENT STATUS (ORDER, PAID, PARTLY
PAID, UNPAID), AMOUNT PAID (=10), YEAR DUE (=YEAR OF TRANSACTION),
MONTH DUE (=MONTH OF TRANSACTION), DAY DUE (=DAY OF TRANSACTION)*
```

In actual practice some of the descriptor names might be shortened to make querying of the bank less verbose (e.g., NATURE OF TRANSACTION might be shortened to TYPE). These names were chosen to make the descriptors roughly self-explanatory.

The bank is now structured but empty. Let us put some data in it.

```
DEFINE ITEMS FROM CARDS, FORMAT FREE 1-3, 7, 6, 4-5, 8-12, 15,
13-14*

14616, CUTRATE FURNITURE CO., 415 KNOTHOLE ST., MAY, 15, 1960,
EXPENSE, 3 DESKS WITH CHAIRS, DELIVERED, 180.79, UNPAID, 0,
JULY, 15, 1960*

14617, ECONOMO DRUGS, 800 BONGO DRIVE, MAY, 15, 1960, 1, 6 VANILLA
+ 2 CHOCOLATE ICE CREAM SANDWICHES, 1, 1.29, PAID, 1.29,,,*

14618, J.W.SWINDLER, 99 CREDIT CARD BLVD., 5, 15, 1960, INCOME,
2 XYZ SPECIALS, 3, 4.99, 2, 1.00, JUNE, 10, 1960*
     :
     ↓
END OF ITEMS
```

The XYZ data bank is now in existence and ready for querying. The next day the XYZ manager decides to experiment with a new data format.

```
DEFINE ITEMS FROM TAPE - FIXED 1-5, 7-80, 2/7-2/30, 3/1, 3/3-3/6,
3/8-3/9, 3/11-3/12, 2/31-2/80, 3/18, 3/20-3/26, 3/28, 3/30-3/36,
3/40-3/43, 3/45-3/46, 3/48-3/49*

14682 ABC CO.
      45 SHMOO AVE.                6 XYZ SPECIALS
2 1960  5 16     1    14.97 1   14.97    1960  6 10

14683 PQR CO.
      47 SHMOO AVE.                2 XYZ SPECIALS
2 1960  5 16     3     4.99 3       0    1960  6 10
     :
     ↓
END OF ITEMS
```

Note that the DEFINE DESCRIPTORS statement is required only once, at the time of the creation of the data bank. After that items may be added as needed by using the Taxir program sequence:

```
DEFINE ITEMS
Item Definition
     :
     ↓
Item Definition
END OF ITEMS
```

To preserve the data bank between runs and bring it back into the machine as needed, see the section on READ DATA BANK and WRITE DATA BANK statements.

---o---

3. QUERYING A TAXIR DATA BANK

STATEMENT TYPE: CONTROL VOCABULARY

Having built a data bank by means of the procedure described in the last chapter, the user is now in a position to address queries to the Taxir system which cause selected information from the data bank to be printed. The user makes these selections by writing Taxir statements containing boolean expressions and these expressions are composed of words and punctuation chosen from a special list called the control vocabulary. Every data bank has its own control vocabulary. Permanent members of the control vocabulary of any and all data banks are the following character strings:

bNOTb
bANDb
bORb
bFROMb
bTOb
bUNKNOWNb
bRESULTb
(
)
,
*

The balance of the control vocabulary for any one data bank consists of the descriptor names and descriptor-state names defined for that bank by the user.

$$\boxed{\text{CONTROL VOCABULARY } \underbrace{\text{FOR d, d,...d}}_{\text{optional}} \text{ *}}$$

The CONTROL VOCABULARY statement (minus the optional portion) generates a printout of this latter part of the control vocabulary, i.e., a list of all descriptor names together with their state names. With this printout available the user has a useful guide for the composing of queries to his data bank.

d = any descriptor name defined for this data bank. This option permits the user to print selected portions of the control vocabulary, i.e., for just the descriptors specified.

Examples: CONTROL VOCABULARY*
CONTROL VOCABULARY FOR FAMILY, GENUS, SPECIES*
CONTROL VOCABULARY FOR NAME OF INSTRUCTOR*

The following statements cause changes in the control vocabulary:

DEFINE DESCRIPTORS
DEFINE MORE DESCRIPTORS
DELETE STATE
Item Definition } when these introduce new state names
CORRECTION } to descriptors defined under the NAME option

In any run in which statements of the above types appear, the user is advised to follow the last of these statements with a CONTROL VOCABULARY statement requesting at least the descriptors affected.

BOOLEAN EXPRESSIONS

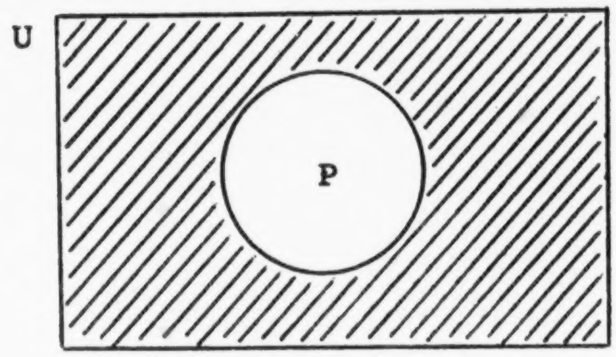
The power to retrieve information selectively from a data bank is the power to name subsets of interest from the total bank. The language of boolean algebra is the language of choice for this purpose. The set whose elements are all the possible subsets of a set P is called the power set of P. If n is the number of elements in P, then the number of elements in the power set of P is 2^n. If a data bank contains 10000 different items, then there are a total of 2^{10000} different subsets that can be specified by boolean algebraic expressions in the Taxir language.

A boolean expression consists of a series of operators and operands. A set of rules defines how the operators act on the operands to yield a result. In ordinary arithmetic expressions, for example, the operators are addition (+), subtraction (-), multiplication (x) and division (÷) and the operands and the result are numbers (or variables like x, y, z, which stand for numbers). In boolean expressions the operators are complement (NOT), intersection (AND), and union (OR). The operands are sets and the result is a set.

Boolean Operators

The three operations of boolean algebra are defined as follows:

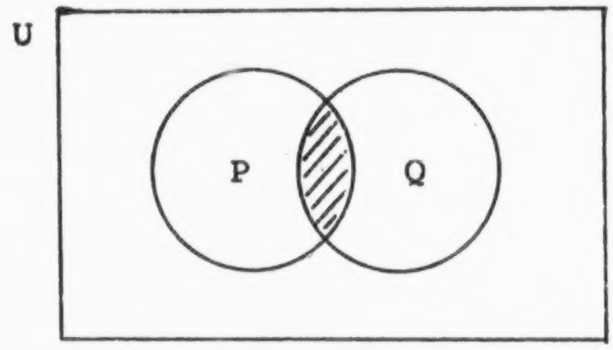
1. The complement of a set P (written NOT P) is the set of all elements in U which are not in P.



Example: U = all the people at the party
P = those with hats
NOT P = those without hats

NOT P is the set shown by the shaded area.

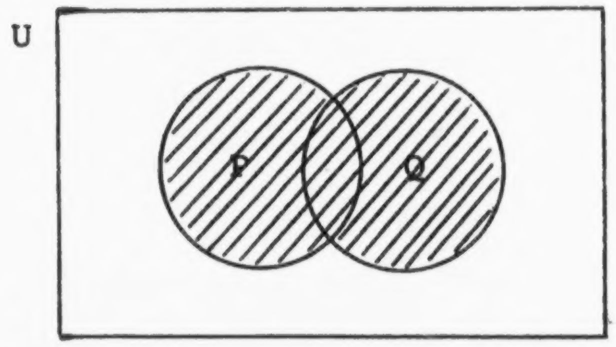
2. The intersection of a set P and a set Q (written P AND Q) is the set of all elements in U which are in both P and Q.



Example: U = all the people at the party
P = those with hats
Q = those with coats
P AND Q = those with both hats and coats

P AND Q is the set shown by the shaded area.

3. The union of a set P and a set Q (written P OR Q) is the set of all elements in U which are in P or Q or both.



Example: U = all the people at the party
P = those with hats
Q = those with coats
P OR Q = those with hats or coats or both

P OR Q is the set shown by the shaded area.

Here follows a list of true statements about any sets P and Q. It is not necessary to learn these, but you may consider it a good sign if your intuition is satisfied of their truth. Remember that U = universal set and ∅ = null set.

If P = U, then NOT P = ∅.
If P = ∅, then NOT P = U.
If P and Q are disjoint, i.e., have no elements in common, then
    P AND Q = ∅.
If P is a subset of Q, then P AND Q = P.
If P is a subset of Q and Q is a subset of P, then P AND Q = P = Q.
P AND U = P.
P AND ∅ = ∅.
P AND P = P.
P AND NOT P = ∅.
If P is a subset of Q, then P OR Q = Q.
P OR U = U.
P OR ∅ = P.
P OR P = P.
P OR NOT P = U.
P AND Q is a subset of P OR Q.

Note that complement operates on a single operand, such as NOT P, while intersection and union operate on 2 operands, such as P AND Q and P OR Q.

Taxir Boolean Operands (Type 1)

In a Taxir boolean expression the basic operand is of the form d,ds where d is any descriptor name chosen from the control vocabulary and ds is the name of one of the states of that descriptor, also chosen from the control vocabulary. Such a pair is always separated by a comma.

Examples: GENUS, ROSA
FLOWER COLOR, RED
YEAR, 1965
MONTH, JUNE
RELATIVE HUMIDITY, 30

The set specified by an operand of the type d,ds is the set of all items in the data bank that have been assigned to the state ds for the descriptor d.

In addition to the descriptor-states defined by the user, each descriptor is automatically assigned by Taxir a state called UNKNOWN. This state may also be used in query operands.

Examples: GENUS, UNKNOWN
FLOWER COLOR, UNKNOWN
YEAR, UNKNOWN
MONTH, UNKNOWN
RELATIVE HUMIDITY, UNKNOWN

In descriptors defined under the FROM-TO option there is a further option to define a label (see p. 12). In such cases it is permissible, but never necessary, to include the label in the query operand.

Example: In a data bank dealing with weather records, one of the descriptors is defined as MAXIMUM TEMPERATURE (FROM -50 TO 110 IN DEGREES F.). Some valid query operands would be:

MAXIMUM TEMPERATURE, 75
MAXIMUM TEMPERATURE, 75 DEGREES F.

Taxir Boolean Operands (Type 2)

In a series of operands connected by OR's such as d,ds OR d,ds OR...OR d,ds, if d is the same for all members of the series, then the user may employ for his convenience a second form of operand:

d,ds OR ds OR...OR ds

Example: Instead of GENUS, ROSA OR GENUS, SPIRAEA OR GENUS, PRUNUS OR GENUS, CRATEAGUS the user may substitute GENUS, ROSA OR SPIRAEA OR PRUNUS OR CRATAEGUS.

No such practice is permitted for series of operands connected by AND's. Since the states of a single descriptor are mutually exclusive, i.e., describe disjoint subsets of the data bank, then d,ds AND d,ds (where d is the same in both operands) must yield the null set. RELATIVE HUMIDITY, 30 AND RELATIVE HUMIDITY, 40 is a legal expression, but yields the null set, as there can be no item in the bank assigned to both those states. Taxir does not permit an operand such as RELATIVE HUMIDITY, 30 AND 40.

Taxir Boolean Operands (Type 3)

A third type of Taxir operand is of the form d, FROM ds1 TO ds2 where d is any descriptor name chosen from the control vocabulary which was defined under the ORDER or FROM-TO options, and dsl and ds2 are the names of two of the states of the descriptor d, also chosen from the control vocabulary. dsl must be earlier in the order already defined than ds2.

Example: Here is part of a DEFINE DESCRIPTORS statement: TEMPERATURE (FROM -50 TO 110 BY 2 IN DEGREES F.), MONTH (ORDER, JAN., FEB., MAR., APR., MAY, JUNE, JULY, AUG., SEPT., OCT., NOV., DEC.), DAY (FROM 1 TO 31), COLOR (ORDER, RED, ORANGE, YELLOW, GREEN, BLUE, VIOLET), COUNTRY (NAME, 100). The following are valid operands chosen from the above bank:

```
TEMPERATURE, FROM 0 TO 10
TEMPERATURE , FROM 0 TO 10 DEGREES F.
TEMPERATURE, FROM 0 DEGREES F. TO 10 DEGREES F.
MONTH, FROM MAY TO AUG.
DAY, FROM 7 TO 9
COLOR, FROM RED TO YELLOW
```

An operand of the type d, FROM dsl TO ds2 is logically equivalent to d, dsl OR a OR b OR c OR...OR ds2, where a, b, c, etc. are all the states between dsl and ds2 in the order defined.

```
Examples: DAY, FROM 7 TO 9 ≡ DAY, 7 OR 8 OR 9
          MONTH, FROM MAY TO AUG. ≡ MONTH, MAY OR JUNE OR JULY OR
                                AUG.
          TEMPERATURE, FROM -4 TO 6 ≡ TEMPERATURE, -4 OR -2 OR
                                0 OR 2 OR 4 OR 6
```

Whereas an operand such as COST, FROM 1 TO 1000000 is logically equivalent to a series of a million descriptor-states connected by OR's, Taxir generates for actual execution a much more efficient equivalent boolean expression. See Estabrook G.F. and R.C. Brill, 1969, The Theory of the Taxir Accessioner, Mathematical Biosciences, 5:327-340.

Still in the framework of the above examples, the following attempts at operands are invalid:

TEMPERATURE, FROM 10 TO 0 (dsl is not earlier in the order defined than ds2.)

MONTH, FROM AUG. TO MAY (Same reason as above.)

TEMPERATURE, FROM 90 TO 114 (One of the states is not in the descriptor, i.e., out of the numeric range defined.)

TEMPERATURE, FROM 90 TO 99 (One of the states is not in the descriptor, i.e., not obtainable by increments of 2 from the initial value defined.)
COLOR, FROM YELLOW TO BROWN (One of the states is not in the descriptor, i.e., not in the list provided in the DEFINE DESCRIPTORS statement.)
COUNTRY, FROM ALBANIA TO DENMARK (No order has been defined under the NAME option.)

The FROM ds1 TO ds2 portion of a type 3 operand may play the role of a ds portion in a type 2 operand.

Examples: YEAR, 1910 OR FROM 1920 TO 1931 OR 1938
MONTH, FROM JAN. TO MAR. OR FROM JUNE TO OCT. OR DEC.
TEMPERATURE, FROM 0 TO 10 OR FROM 30 TO 60 OR FROM 80 TO 100

Taxir Boolean Operands (Type 4)

Finally there is a fourth type of operand, which is simply the word RESULT. This refers to the subset of the data bank specified by the boolean expression of the previously posed query.

Combining Operators and Operands

The rules for combining operators and operands into boolean expressions now follow:

1. If P is an operand, then P is a boolean expression.

   All the examples of operands shown in the previous sections are also examples of boolean expressions.

2. If P and Q are boolean expressions, then NOT (P), (P) AND (Q), and (P) OR (Q) are boolean expressions.

   Examples: NOT (GENUS, ROSA)
   (GENUS, ROSA) AND (YEAR OF COLLECTION, 1910)
   (GENUS, ROSA) OR (LEAF LENGTH, FROM 10 TO 30 MM.)

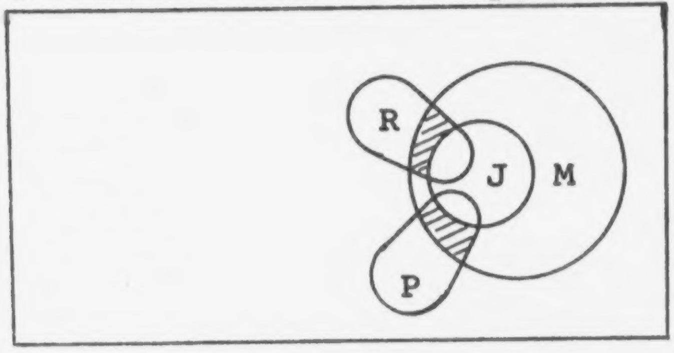
   This rule implies that boolean expressions may themselves be operands in yet larger boolean expressions.

   Examples: ((FLOWER COLOR, RED OR PINK) AND (COUNTRY OF COLLECTION, MEXICO)) AND (NOT (PROVINCE OF COLLECTION, JALISCO))

   ((SEMESTER, SPRING) AND (DEPT., BIOLOGY)) AND (INSTRUCTOR, JONES OR CARTER)

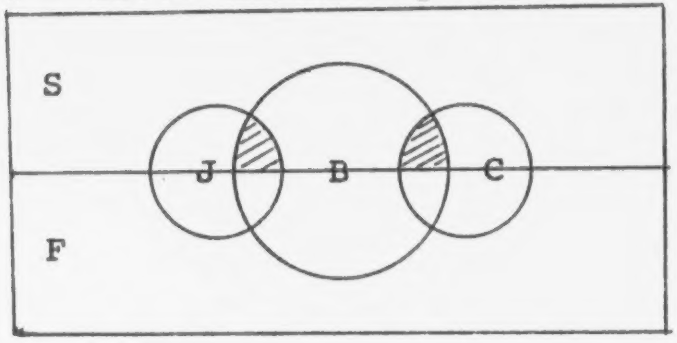
The subsets specified by the boolean expressions in the above two examples are shown by the shaded areas in the diagrams below:

U = 100000 herbarium specimens



R = FLOWER COLOR, RED
P = FLOWER COLOR, PINK
M = COUNTRY OF COLLECTION, MEXICO
J = PROVINCE OF COLLECTION, JALISCO

U = 12000 university courses



S = SEMESTER, SPRING
F = SEMESTER, FALL
B = DEPT., BIOLOGY
J = INSTRUCTOR, JONES
C = INSTRUCTOR, CARTER

3. If P is a boolean expression, then P, (P), ((P)), (((P))), etc. are equivalent boolean expressions.

Examples: TEMPERATURE, 50 = (TEMPERATURE, 50) = ((TEMPERATURE, 50))
TEMPERATURE, 50 AND HUMIDITY, 45 = (TEMPERATURE, 50 AND HUMIDITY, 45) = ((TEMPERATURE, 50 AND HUMIDITY, 45)) = ((TEMPERATURE, 50 AND (HUMIDITY, 45)))

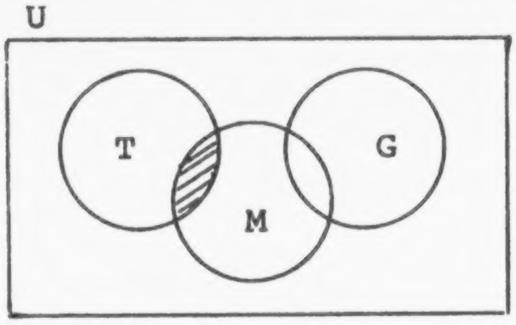
Two or more pairs of parentheses enclosing a boolean expression are permitted, but are always redundant. A single pair of parentheses is sometimes redundant and sometimes essential.

The Use of Parentheses in Boolean Expressions

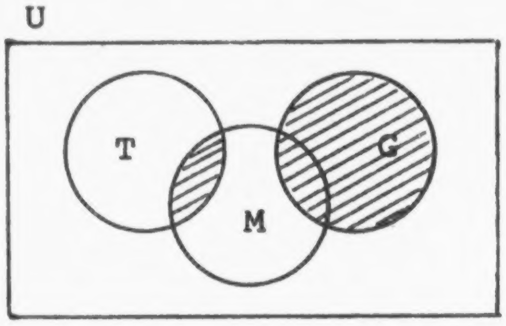
In the examples illustrating rule 2 above the parentheses reflect the manner in which boolean expressions are built up from smaller boolean expressions by the recursive power of the rule. In all such cases, the innermost pairs of parentheses enclose boolean expressions that may be simple operands of type 1 through 4. The next, more outward, level of parentheses may enclose more complex expressions that include one operator. The levels may proceed outward indefinitely, introducing new operators and more complex expressions. During query execution, such complex expressions must have their internal operations performed before they can serve as operands in more outward level expressions. So parentheses indicate the order of operations in boolean expressions. The operations called for within innermost parentheses are performed first, thereafter operations proceed outwards.

Examples: (CUSTOMER, TOLSTOY L. AND MONTH OF SALE, MARCH) OR
CUSTOMER, GOGOL N.
CUSTOMER, TOLSTOY L. AND (MONTH OF SALE, MARCH OR
CUSTOMER, GOGOL N.)

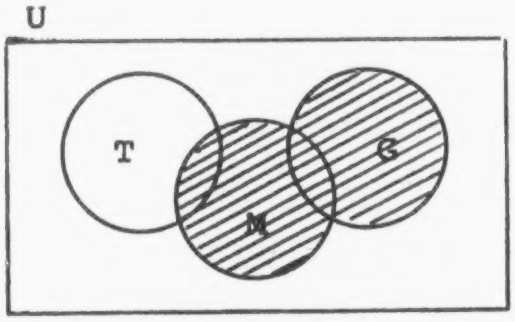
In the first example the AND is performed first on its neighbor operands, then the result of that operation is OR'd with CUSTOMER, GOGOL N.. In the second example the OR is performed first on its neighbor operands, then the result of that operation is AND'd with CUSTOMER, TOLSTOY L.. The sets specified by these two boolean expressions are not the same. The first expression yields the set of Tolstoy's March purchases and all of Gogol's purchases. The second expression yields only Tolstoy's March purchases. If Gogol has bought even one bottle of hair spray these cannot be the same sets. U = purchases at Economo Drugs. T = Tolstoy's purchases. G = Gogol's purchases. M = March purchases. P = intermediate result. Q = final result.



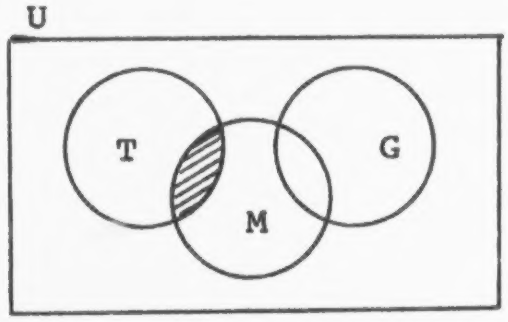
(CUSTOMER, TOLSTOY L. AND
MONTH OF SALE, MARCH) = P(shaded)



→ P OR CUSTOMER, GOGOL N. =
Q(shaded)



(MONTH OF SALE, MARCH OR
CUSTOMER, GOGOL N.) = P(shaded)



→ P AND CUSTOMER, TOLSTOY L. =
Q(shaded)

Parentheses are needed to resolve ambiguity over the desired order of operations. However, there are some other rules, about to be stated, which permit resolution of ambiguity without parentheses, and in fact, most typical Taxir queries will not need parentheses at all.

In the absence of parentheses Taxir follows a built-in priority of operations. The first operations performed are those called for by simple operands of types 2 or 3.

Examples: NOT FLOWER COLOR, RED OR BLUE = NOT (FLOWER COLOR, RED OR BLUE)
MONTH, FROM JAN. TO MAR. OR JULY AND YEAR, FROM 1920 TO 1930 = (MONTH, FROM JAN. TO MAR. OR JULY) AND (YEAR, FROM 1920 TO 1930)
DEPT., BIOLOGY OR MICROBIOLOGY OR INSTRUCTOR, JONES OR CARTER = (DEPT., BIOLOGY OR MICROBIOLOGY) OR (INSTRUCTOR, JONES OR CARTER)

We distinguish here between two kinds of OR's. The OR of an operand type 2 takes priority over the OR connecting two boolean expressions. After the operations called for by simple operands, the built-in priority continues in the order: NOT's, then AND's, then OR's.

Examples: NOT WAREHOUSE, A AND DISTRICT, 4 = (NOT WAREHOUSE, A) AND DISTRICT, 4
NOT SHELF, 7 OR CABINET, A22 = (NOT SHELF, 7) OR CABINET, A22
GENUS, ROSA AND YEAR, 1910 OR LEAF LENGTH, 40 = (GENUS, ROSA AND YEAR, 1910) OR LEAF LENGTH, 40
NOT COLOR, GREEN AND NOT SIZE, MEDIUM OR NOT STYLE, 3450 = ((NOT COLOR, GREEN) AND (NOT SIZE, MEDIUM)) OR (NOT STYLE, 3450)

Parentheses are used to indicate that the operators within parentheses are to operate before those immediately outside to right or left. Therefore, it is never necessary to enclose simple operands of types 1 or 4, because they have no operators. It is never necessary to enclose simple operands of types 2 or 3, because their operators always operate first by rules of priority. It is never necessary to enclose a complete boolean expression, because there are no operators outside.

It is necessary to enclose only sub-expressions containing at least one operator, and then only when at least one of the operators inside is of lower priority than those immediately outside to right or left. If the operators inside are of higher priority than those immediately outside, then the parentheses are in agreement with the rules of priority and are not needed. If the operators inside are of the same priority with those immediately outside, again the parentheses are redundant, as the order of operations at the same priority level does not affect the result.

Examples: YEAR, 1970 AND MONTH, SEPT. AND DAY, 3 = (YEAR, 1970 AND MONTH, SEPT.) AND DAY, 3 = YEAR, 1970 AND (MONTH, SEPT. AND DAY, 3)
AGE, FROM 18 TO 22 OR EYESIGHT, 20/20 OR SAFETY RATING, A1 = (AGE, FROM 18 TO 22 OR EYESIGHT, 20/20) OR SAFETY RATING, A1 = AGE, FROM 18 TO 22 OR (EYESIGHT, 20/20 OR SAFETY RATING, A1)
NOT NOT STYLE, H = NOT (NOT STYLE, H) = STYLE, H
NOT NOT NOT STYLE, H = NOT (NOT (NOT STYLE, H)) = NOT STYLE, H

Not only are no parentheses needed for series of NOT's, but the series of NOT's are themselves redundant, as all such series reduce either to P or to NOT P.

The priority of an operand type 2 or 3 cannot be overridden by parentheses. Expressions such as (NOT FLOWER COLOR, RED) OR PINK are invalid. Therefore, it is never necessary to enclose in parentheses a sub-expression whose only operator is NOT.

This leaves but a few situations where parentheses are essential:

To give AND priority over NOT, as in NOT (BUILDING, OLD MAIN AND ROOM, 100).

To give OR priority over NOT, as in NOT (LENGTH, 40 FT. OR WIDTH, 40 FT.). Or as in NOT (HEIGHT, 10 FT. AND LENGTH, 40 FT. OR WIDTH, 40 FT.).

To give OR priority over AND, as in CUSTOMER, TOLSTOY L. AND (MONTH OF SALE, MARCH OR CUSTOMER, GOGOL N.). Or as in (CUSTOMER, TOLSTOY L. AND MONTH OF SALE, MARCH OR CUSTOMER, GOGOL N.) AND ARTICLE PURCHASED, HAIR SPRAY.

This section ends with an example illustrating most of the topics discussed in this section. If this example makes sense, then you know all you need to know about parentheses in Taxir boolean expressions. All parentheses shown here are needed.

Example: In a data bank in which each item is associated with a date, that is, with the descriptors YEAR, MONTH, DAY, we wish to specify those items which fall into the continuous daily time period from Oct. 7, 1960 to May 25, 1964, but excluding the dates April 1, 2, 3, 4, 5, and 7, 1963.

```
(YEAR, 1960 AND (MONTH, OCT. AND DAY, FROM 7 TO 31 OR MONTH, NOV.
OR DEC.) OR YEAR, FROM 1961 TO 1963 OR YEAR, 1964 AND (MONTH, FROM
JAN. TO APR. OR MONTH, MAY AND DAY, FROM 1 TO 25)) AND NOT (YEAR,
1963 AND MONTH, APR. AND DAY, FROM 1 TO 5 OR 7)
```

STATEMENT TYPE: HOW MANY

There are 5 statement types in the Taxir language that call for the use of boolean expressions. We are now ready to describe the simplest of these. Its general format is:

$$\boxed{\text{HOW MANY noise } \begin{Bmatrix}\text{bWITHb}\\ \text{bHAVEb}\end{Bmatrix} \text{ boolean expression*}}$$

noise = any combination of characters except the words bWITHb or bHAVEb. Noise may be omitted, but the noise terminator (bWITHb or bHAVEb) must appear.

Examples:

```
HOW MANY SPECIMENS IN THE MUSEUM HAVE FLOWER COLOR, RED*
HOW MANY COURSES ARE THERE WITH DEPT., HISTORY AND BUILDING, HELLEMS*
HOW MANY EMPLOYEES WITH MARITAL STATUS, MARRIED AND SEX, MALE AND
     NOT DEPENDENTS, 0*
HOW MANY WITH RESULT AND SALARY, FROM 8000 TO 10000*
```

The system responds to such queries by printing 3 lines of information, answering the question how many, as well as stating what percentage of the total bank this represents.

Example:

```
NO. OF ITEMS IN QUERY RESPONSE = 172
NO. OF ITEMS IN THE DATA BANK = 11535
PERCENTAGE OF RESPONSE/TOTAL DATA BANK = 1.49
```

STATEMENT TYPE: PRINT

This statement type is the most powerful of the Taxir query statements. Its general format is:

PRINT noise: descriptor list bFORb noise $\left\{\begin{matrix}\text{bWITHb}\\ \text{bHAVEb}\end{matrix}\right\}$ boolean expression*

The system responds with the same 3 line printout as it does to the HOW MANY statement. In addition the system prints the states of whichever descriptors the user selects (in the descriptor list) for each item that is a member of the set specified by the boolean expression.

In the first instance noise = any string of characters except colon (:). In the second instance noise = any string of characters except the words bWITHb or bHAVEb. Noise may be omitted, but the noise terminator (colon in the first instance and bWITHb or bHAVEb in the second instance) must appear.

Descriptor Lists

In its simplest form a descriptor list is a series of descriptor names chosen from the control vocabulary and separated by commas. The list may have from one to as many members as there are descriptors defined for the bank. If the list has one member, no commas are used.

Examples: NAME OF CUSTOMER, ACCOUNT NO., DATE OF TRANSACTION
MONTH, DAY, YEAR
COURSE NO., BUILDING, ROOM NO., STARTING TIME, INSTRUCTOR
ROOM NO.
FAMILY, GENUS, SPECIES
GENUS

For each item in the subset specified by the boolean expression, the system will print the appropriate descriptor-state for each of the descriptors in the list. In the simplest case of a one member descriptor list, the printout will be nothing more than an ordered list of states for that descriptor with duplicates omitted. It will, in fact, be a subset of the control vocabulary and will be ordered as it is. If the descriptor was earlier defined (in the

DEFINE DESCRIPTORS statement) under the NAME option, the order will be alphabetic. If it was defined under the FROM-TO option, the order will be numeric and if it was defined under the ORDER option, the order will be that specified in the DEFINE DESCRIPTORS statement. In all cases, the UNKNOWN state (which is printed as 3 dashes) occurs last in the printout order.

Example: In a data bank where the items are university courses the descriptor list in a query might be the single descriptor, DEPARTMENT, previously defined under the NAME option. Query printout will be alphabetically ordered and might look like this:

```
AEROSPACE ENGINEERING
ASTROGEOPHYSICS
BIOLOGY
EDUCATION
FINE ARTS
GEOLOGY
MATHEMATICS
ZOOLOGY
```

Example: In the same data bank the descriptor list for another query might be the single descriptor, COURSE LEVEL, previously defined as follows: COURSE LEVEL (ORDER, UNDERGRAD, GRAD, SPECIAL). Query printout will follow the order defined:

```
UNDERGRAD
GRAD
SPECIAL
```

If the descriptor list has more than one member, and this is the typical case, the printout will be ordered hierarchically and indented to reveal this hierarchical structure. The first descriptor (left-most) in the list will be assigned the highest order in the hierarchy, the second descriptor in the list will be assigned the second highest order, and so forth.

Example: In the same bank of university courses the descriptor list for a query might be DEPARTMENT, COURSE LEVEL, COURSE NUMBER. The printout would look like this (if the boolean expression happened to specify the following 19 items):

```
AEROSPACE ENGINEERING
     UNDERGRAD
          100
          101
          105
     GRAD
          500
          501
          510
     SPECIAL
          150
          160
BIOLOGY
     UNDERGRAD
          105
          110
     GRAD
          500
          512
          513
CHEMISTRY
     UNDERGRAD
          100
          111
          152
          175
     SPECIAL
           40
           41
```

Example: A query with the same boolean expression but with the descriptor list permuted to COURSE LEVEL, DEPARTMENT, COURSE NUMBER would generate the following printout:

```
UNDERGRAD
     AEROSPACE ENGINEERING
          100
          101
          105
     BIOLOGY
          105
          110
     CHEMISTRY
          100
          111
          152
          175
GRAD
     AEROSPACE ENGINEERING
          500
          501
          510
     BIOLOGY
          500
          512
          513
SPECIAL
     AEROSPACE ENGINEERING
          150
          160
     CHEMISTRY
           40
           41
```

Example: The same boolean expression and yet another permutation of the descriptor list to COURSE NUMBER, COURSE LEVEL, DEPARTMENT would generate the following printout:

```
 40
     SPECIAL
          CHEMISTRY
 41
     SPECIAL
          CHEMISTRY
100
     UNDERGRAD
          AEROSPACE ENGINEERING
          CHEMISTRY
101
     UNDERGRAD
          AEROSPACE ENGINEERING
105
     UNDERGRAD
          AEROSPACE ENGINEERING
          BIOLOGY
110
     UNDERGRAD
          BIOLOGY
111
     UNDERGRAD
          CHEMISTRY
150
     SPECIAL
          AEROSPACE ENGINEERING
152
     UNDERGRAD
          CHEMISTRY
160
     SPECIAL
          AEROSPACE ENGINEERING
175
     UNDERGRAD
          CHEMISTRY
500
     GRAD
          AEROSPACE ENGINEERING
          BIOLOGY
501
     GRAD
          AEROSPACE ENGINEERING
510
     GRAD
          AEROSPACE ENGINEERING
512
     GRAD
          BIOLOGY
513
     GRAD
          BIOLOGY
```

Notice in the three preceding examples that the number of lines printed in each example is not the same, even though the same descriptor-states for the same set of 19 items are represented. As long as the system is printing the states of a descriptor which fall hierarchically under the same state of another descriptor, it is unnecessary to repeat the name of the state of higher order. In the first of the three examples:

```
AEROSPACE ENGINEERING
     UNDERGRAD
          100
```

represents a single item. The next item could be represented as:

```
AEROSPACE ENGINEERING
     UNDERGRAD
          101
```

but it saves two print lines and provides a more lucid printout to represent both items as:

```
AEROSPACE ENGINEERING
     UNDERGRAD
          100
          101
```

The differences in the number of lines printed in the three examples arise then, not from differences in the items or descriptors selected, but from differences in the memberships of the different hierarchies established by permuting the descriptor list. The rule then is that before each item is printed it is compared state for state with the previous item printed. Duplicate states are omitted until the first difference occurs. All states from the first difference onward are printed. The only exception to this rule is when an item falls at the top of a new page, in which case the entire item is printed so that reference need not be made to a previous page in picking up the names of duplicate states. (Page break never occurs in the middle of an item.)

Line Combos

In the above illustrations each descriptor-state is printed on a line of its own. The user has an option to cause as many descriptor-states to be printed on the same line as will fit. When any series of contiguous descriptor names in the descriptor list is enclosed in parentheses, their states will appear on the same line in the query printout.

Examples: The descriptor list ALPHA, BETA, GAMMA, EENY, MEENY, MINY, MO will generate printout in the following format:

```
ALPHA
     BETA
          GAMMA
               EENY
                    MEENY
                         MINY
                              MO
```

But the descriptor list (ALPHA, BETA), GAMMA, (EENY, MEENY, MINY, MO) will alter that format to:

```
ALPHA BETA
     GAMMA
          EENY MEENY MINY MO
```

The descriptor list ALPHA, (BETA, GAMMA), (EENY, MEENY), (MINY, MO) will generate printout in the following format:

```
ALPHA
     BETA GAMMA
          EENY MEENY
               MINY MO
```

The descriptor list (ALPHA, BETA, GAMMA) will generate printout in the following format:

```
ALPHA BETA GAMMA
```

The rule on omission of duplicate states is the same in line combos, except that the line combo is treated as though it were a single state. The whole line combo will be printed or omitted, never a portion only.

Example: Under the control of the descriptor list STATE, CITY we might see the following printout:

```
COLORADO
     BOULDER
     DENVER
MASSACHUSETTS
     BOSTON
     WORCESTER
```

Under the control of the descriptor list (STATE, CITY) the same items would appear as:

```
COLORADO      BOULDER
COLORADO      DENVER
MASSACHUSETTS BOSTON
MASSACHUSETTS WORCESTER
```

One use of the line combo option is to place in closer association a series of related descriptors. The above example illustrates this, as does the following:

Example: The descriptor list MONTH, DAY, YEAR might generate the following:

```
MAY
     14
          1962
JUNE
     22
          1962
```

But the descriptor list (MONTH, DAY, YEAR) would yield instead:

```
MAY  14 1962
JUNE 22 1962
```

Line combos save print lines, yielding a more compact (i.e., cheaper) printout. This is especially valuable in long book-length printouts. Remember that a query can cause an entire data bank to be printed and in many applications it is quite useful to generate such a book periodically. In such cases the line combo option can be used to advantage even if the descriptors combined have no special relation to each other.

It can also be used to permit more descriptors in the printout than could otherwise fit. The Taxir system will print under the control of any valid descriptor list, but will truncate at the right-hand margin any information that cannot fit on the print line.

There are 135 character positions in a print line. Under the control of a descriptor list A, B, C, D, E, F, G, H, I, J, K the states of K will begin in character position 51 on the page. This is a result of the indentation rule, which prints each state of the highest order descriptor at the left-hand margin and indents each print line of lower order an additional 5 character positions. If K has any states longer that 85 characters they will be truncated. If line combos are used, the number of print lines and hence the length of indentation of the last print line are reduced. The descriptor list (A, B, C, D), (E, F, G, H), I, J, K, for instance, will cause K to start in character position 21 instead of 51, leaving plenty of room for the long states of K.

On the other hand, the use of line combos can cause line truncation. If in the case above the states of E, F, G and H are long their combination can exceed the limits of the print line.

Print Fields

All descriptor-states printed which belong to the same descriptor will line up on the same character position, either all starting in the same character position (left-justified) or all ending in the same character position (right-justified). This rule holds whether the descriptor is a member of a line combo or not. Descriptors whose states are known to be numbers (i.e., which have been defined under the FROM-TO option) are right-justified. All other descriptors are left-justified.

Example: The descriptor list is (FRUIT, AMOUNT). FRUIT has been defined under the NAME option, AMOUNT has been defined under the FROM-TO option. The printout might be:

```
APPLES     2
BANANAS   14
ORANGES   17
PEARS    952
```

Remember that descriptors defined under the FROM-TO option may have a label and that this label always appears as part of the state in query printouts.

Example: LENGTH is a descriptor defined as follows: LENGTH (FROM 0 TO 10000 IN FT.). Printout might be:

```
   1 FT.
   2 FT.
  18 FT.
 300 FT.
4575 FT.
```

Taxir keeps track while data banks are being built or corrected of the number of characters in the longest state of each descriptor. This information is printed as part of the control vocabulary to help the user in calculating printout fits. When query printout is being prepared by the system, a print field is reserved for each descriptor in the descriptor list, its length being the value just mentioned.

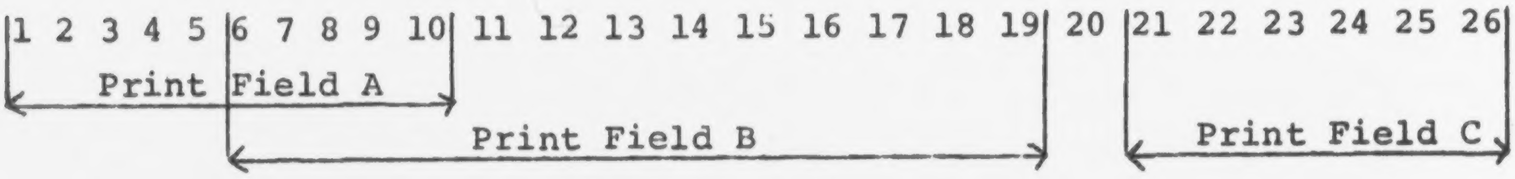
Example: Descriptor list = A, (B,C). The number of characters in the longest states of A = 10, B = 14 and C = 6.

The print field for descriptor A will be character positions 1 through 10. If A is numeric its states will be right-justified on position 10. Otherwise, they will be left-justified on position 1.

The print field for descriptor B will be character positions 6 through 19. If B is numeric its states will be right-justified on position 19. Otherwise, they will be left-justified on position 6.

Position 20 will be blank to guarantee at least one space between the states of B and C.

The print field for descriptor C will be character positions 21 through 26. If C is numeric its states will be right-justified on position 26. Otherwise, they will be left-justified on position 21.



The minimum print field length is 3 character positions. This is because all descriptors have the missing information (UNKNOWN) state, which appears in query printout as 3 dashes. The maximum print field length is 90, as this is also the maximum on descriptor-state length.

Example: Descriptor list = NAME OF EMPLOYEE, (SOCIAL SECURITY NO., YEAR OF BIRTH, MONTH OF BIRTH, DAY OF BIRTH, SEX, MARITAL STATUS, NO. OF DEPENDENTS). The number of characters in the longest states of NAME OF EMPLOYEE = 20, SOCIAL SECURITY NO. = 11, YEAR OF BIRTH = 4, MONTH OF BIRTH = 5, DAY OF BIRTH = 3, SEX = 3, MARITAL STATUS = 8, NO. OF DEPENDENTS = 3. Print field rules will cause printout to appear as follows:

```
ALLEN R. H.
     134-72-1539 1951 MAY     13 M   SINGLE     0
AXELROD G. D.
     175-14-9326 1940 JULY     3 M   MARRIED    2
BAKER J. L.
     922-54-6693 1948 SEPT.   15 F   DIVORCED   1
```

Print Field Override Options (L and R Parameters)

Taxir provides the user additional flexibility in controlling the printing formats of query printouts. The user can override the rules for print fields described above and specify exactly on which character positions he wants his descriptors to be left- or right-justified. He does so by including an override parameter in parentheses immediately following each descriptor in the descriptor list he wishes to so control. The parameter consists of the letter L (for left-justify) or R (for right-justify), followed by the number of the desired character position (must be in the range 1 to 135).

Example: In the previous example the printout appeared crowded and confusing. This can be improved by altering the descriptor list as follows: NAME OF EMPLOYEE, (SOCIAL SECURITY NO.(L15), YEAR OF BIRTH(L33), MONTH OF BIRTH, DAY OF BIRTH, SEX(L54), MARITAL STATUS, NO. OF DEPENDENTS). The printout now looks like this:

```
ALLEN R. H.
              134-72-1539       1951 MAY    13       M    SINGLE     0
AXELROD G. D.
              175-14-9326       1940 JULY    3       M    MARRIED    2
BAKER J. L.
              922-54-6693       1948 SEPT.  15       F    DIVORCED   1
```

Example: Descriptor list = WEATHER STATION(L9), (MONTH(R4), DAY(L6), MINIMUM TEMPERATURE(R12), MAXIMUM TEMPERATURE, MINIMUM RELATIVE HUMIDITY(R24), MAXIMUM RELATIVE HUMIDITY).

```
        MORGAN CREEK
 JAN 9   -50  10      30  35
JULY 10   55  76      20  27
        SUMMIT RIDGE
 JAN 9   -40   7      32  36
JULY 10   61  72      21  30
```

Notice that the amount of indentation may be controlled as well as the length of the print fields and that descriptors which are ordinarily left-justified can be right-justified and vice-versa. Descriptors without override parameters (such as MAXIMUM RELATIVE HUMIDITY above) are subject to the ordinary rules of indentation, justification and print field length described earlier.

In general the override options are used to create more blank space around the printed matter to improve readability. It is possible, however, to reduce the length of a print field. A look at the control vocabulary may show that in one descriptor the length of the longest state is 85 characters, but that the second longest state is only 40 characters long. It may be desirable to reduce the length of the print field to 40 in order to make a line combo fit without truncation, especially if it is known that the 85 character state will not show up in the query response. If it does show up it will be truncated to 40 characters, which will probably leave enough to identify it. Any descriptor-state which exceeds a reduced print field length is truncated to fit, at the right if it is left-justified, at the left if it is right-justified.

Print Field Override Options (F Parameter)

Taxir provides one additional override option. The rule that each descriptor lines up on a character position, left- or right-justified, can be overridden for any member of a line combo but the first. In this way the members of a line combo can be made to follow one another with some specified amount of space between them. The parameter is an F (for follow) followed by the number of blanks desired between the last character of the previous member of the line combo and the first character of this one (must be in the range 0 to 133).

<u>Example</u>: Descriptor list = (MONTH, DAY, YEAR). Printout will be in the format below:

```
JANUARY      1 1910
FEBRUARY    13 1912
MAY          6 1915
```

But with the descriptor list modified to: (MONTH, DAY(F1), YEAR(F1)), the printout changes to:

```
JANUARY 1 1910
FEBRUARY 13 1912
MAY 6 1915
```

Example: Descriptor list = (LAST NAME, FIRST NAME, MIDDLE NAME). This might yield something like this:

```
BURNETT                                       IVY        COMPTON
MACPHERSON                                    AMY        SEMPLE
THOMAS                                        JOHN       CHARLES
```

But with the descriptor list modified to: (LAST NAME, FIRST NAME(F4), MIDDLE NAME(F1)), the printout will be:

```
BURNETT    IVY COMPTON
MACPHERSON    AMY SEMPLE
THOMAS    JOHN CHARLES
```

Example: Descriptor list = (CODE PREFIX, CODE NO.)

```
AB  452
ANK 399
TX   21
Z     4
```

But with this modification: (CODE PREFIX, CODE NO.(F0))

```
AB452
ANK399
TX21
Z4
```

Any F parameter associated with a solo descriptor or with the first member of a line combo will be ignored by the system and the normal rules will apply.

SAME Option

Finally the user may substitute for any of the descriptor lists described above the word SAME. This will cause the descriptor list of the previous query to be applied again for this query.

Example: PRINT: ALPHA, BETA, GAMMA FOR ITEMS WITH boolean expression #1* PRINT: SAME FOR ITEMS WITH boolean expression #2* The printout for both queries will be in the same format.

HOW TO QUERY

Here follow some hints and examples dealing with both statement types HOW MANY and PRINT. Their general formats are repeated below:

HOW MANY noise $\left\{\begin{matrix}\text{bWITHb}\\\text{bHAVEb}\end{matrix}\right\}$ boolean expression*

PRINT noise: descriptor list bFORb noise $\left\{\begin{matrix}\text{bWITHb}\\\text{bHAVEb}\end{matrix}\right\}$ boolean expression*

Examples: The following examples present some simple solutions to problems that might arise in a marketing situation. The items in this hypothetical data bank are units sold. A customer comes in with Gidget #100425. It is broken and he has lost his sales slip. Is the 90-day warranty still in effect?

```
PRINT: (MONTH OF SALE, DAY OF SALE, YEAR OF SALE) FOR ITEMS WITH
ARTICLE, GIDGET AND SERIAL NO., 100425*

NO. OF ITEMS IN QUERY RESPONSE = 1
NO. OF ITEMS IN THE DATA BANK = 5310
PERCENTAGE OF RESPONSE/TOTAL DATA BANK = 0.02

SEPT. 27 1969
```

What color widgets sold best last summer?

```
HOW MANY UNITS WERE SOLD WITH ARTICLE, WIDGET AND YEAR OF SALE,
1969 AND MONTH OF SALE, FROM JUNE TO AUG. AND COLOR, RED*

NO. OF ITEMS IN QUERY RESPONSE = 5
NO. OF ITEMS IN THE DATA BANK = 5310
PERCENTAGE OF RESPONSE/TOTAL DATA BANK = 0.09

HOW MANY UNITS WERE SOLD WITH ARTICLE, WIDGET AND YEAR OF SALE,
1969 AND MONTH OF SALE, FROM JUNE TO AUG. AND COLOR, WHITE*

NO. OF ITEMS IN QUERY RESPONSE = 3
NO. OF ITEMS IN THE DATA BANK = 5310
PERCENTAGE OF RESPONSE/TOTAL DATA BANK = 0.06

HOW MANY UNITS WERE SOLD WITH ARTICLE, WIDGET AND YEAR OF SALE,
1969 AND MONTH OF SALE, FROM JUNE TO AUG. AND COLOR, BLUE*

NO. OF ITEMS IN QUERY RESPONSE = 7
NO. OF ITEMS IN THE DATA BANK = 5310
PERCENTAGE OF RESPONSE/TOTAL DATA BANK = 0.13
```

How does that break down by month and who were the salesmen involved?

```
PRINT THE FOLLOWING: COLOR, MONTH OF SALE, (SERIAL NO., SALESMAN)
FOR UNITS WITH ARTICLE, WIDGET AND YEAR OF SALE, 1969 AND MONTH OF
SALE, FROM JUNE TO AUG. AND COLOR, RED OR WHITE OR BLUE*
```

```
NO. OF ITEMS IN QUERY RESPONSE = 15
NO. OF ITEMS IN THE DATA BANK = 5310
PERCENTAGE OF RESPONSE/TOTAL DATA BANK = 0.28

BLUE
     JUNE
          101772 IKE
          103428 IKE
          103589 MIKE
     JULY
          106675 JOE
          107781 MOE
     AUG.
          107919 IKE
          108832 JOE
RED
     JUNE
          102445 JOE
          102787 MOE
          103481 IKE
     JULY
          105334 MIKE
          106622 MIKE
WHITE
     JUNE
          101355 JOE
     AUG.
          107445 IKE
          108916 MOE
```

How did Ike do in the first quarter of this year?

```
HOW MANY UNITS WERE SOLD WHICH HAVE SALESMAN, IKE AND YEAR OF SALE,
1970 AND MONTH OF SALE, FROM JAN. TO MAR.*

NO. OF ITEMS IN QUERY RESPONSE = 22
NO. OF ITEMS IN THE DATA BANK = 5310
PERCENTAGE OF RESPONSE/TOTAL DATA BANK = 0.41
```

How does that break down?

```
PRINT A LIST OF: ARTICLE, COLOR, (MONTH OF SALE, DAY OF SALE, SERIAL
NO.(L25)) FOR ITEMS WITH RESULT*

NO. OF ITEMS IN QUERY RESPONSE = 22
NO. OF ITEMS IN THE DATA BANK = 5310
PERCENTAGE OF RESPONSE/TOTAL DATA BANK = 0.41

FIDGET
     BLACK
          JAN.    3      149723
          JAN.   17      153209
          FEB.    7      158334
          FEB.    7      158335
          FEB.   15      167444
          MAR.   21      176550
     BROWN
          JAN.   29      155001
          JAN.   31      156499
          FEB.    8      159900
GIDGET
     GREEN
          JAN.   15      100779
          JAN.   17      100893
          MAR.   22      100998
     PURPLE
          JAN.   11      100565
          JAN.   12      100578
          FEB.    1      100672
          FEB.    3      100675
          MAR.    4      100877
WIDGET
     BLUE
          JAN.   22      456090
          JAN.   24      456098
     RED
          JAN.    7      422989
          MAR.   30      488722
     WHITE
          FEB.   24      467023
```

Examples: The following examples deal with a data bank keeping track of the specimens in a botanical museum. A worker wants to know what specimens the museum has of a certain genus that were collected during a series of expeditions in the 1930's.

```
PRINT: CATALOG NO. FOR SPECIMENS WITH GENUS, MANIHOT AND YEAR OF
COLLECTION, FROM 1932 TO 1938*

NO. OF ITEMS IN QUERY RESPONSE = 8
NO. OF ITEMS IN THE DATA BANK = 103170
PERCENTAGE OF RESPONSE/TOTAL DATA BANK = 0.01

624420
650063
687552
688105
710429
724487
725800
753216
```

These catalog numbers are enough for him to retrieve the actual specimens from the herbarium, provided the museum curator has arranged the specimens in numeric order on the shelves. In many museums the specimens are arranged alphabetically by geographical location and within geographical location alphabetically by taxonomic name and within taxonomic name numerically by catalog number. In such a museum the user might request:

```
PRINT: COUNTRY OF COLLECTION, TAXONOMIC NAME, CATALOG NO. FOR
SPECIMENS WITH GENUS, MANIHOT AND YEAR OF COLLECTION, FROM 1932
TO 1938*

NO. OF ITEMS IN QUERY RESPONSE = 8
NO. OF ITEMS IN THE DATA BANK = 103170
PERCENTAGE OF RESPONSE/TOTAL DATA BANK = 0.01

BR. HONDURAS
     MANIHOT GUALANENSIS
          753216
HONDURAS
     MANIHOT AESCULIFOLIA
          687552
MEXICO
     MANIHOT ESCULENTA
          624420
          688105
          724487
          725800
     MANIHOT PARVICOCCA
          650063
     MANIHOT PRINGLEI
          710429
```

The worker may not need to look at specimens. There may be enough information in the data bank to answer his questions. Perhaps his principal interest is in the collectors' names and collecting dates of the specimens in question, all of which are descriptors in the bank. In this case he might ask:

```
PRINT: COLLECTOR, (YEAR OF COLLECTION, MONTH OF COLLECTION(F1), DAY
OF COLLECTION(F1)), (CATALOG NO., TAXONOMIC NAME, COUNTRY OF COLLEC
TION, PROVINCE OF COLLECTION) FOR SPECIMENS WITH GENUS, MANIHOT AND
YEAR OF COLLECTION, FROM 1932 TO 1938*

NO. OF ITEMS IN QUERY RESPONSE = 8
NO. OF ITEMS IN THE DATA BANK = 103170
PERCENTAGE OF RESPONSE/TOTAL DATA BANK = 0.01

BANGHAM W.N.
     1935 AUG. 10
          687552 MANIHOT AESCULIFOLIA    HONDURAS      CORTES
GENTLE P.H.
     1938 MAY 1
          753216 MANIHOT GUALANENSIS     BR. HONDURAS  EL CAYO
HINTON G.B.
     1933 JUNE 21
          624420 MANIHOT ESCULENTA       MEXICO        SAN LUIS POTOSI
     1936 JULY 7
          710429 MANIHOT PRINGLEI        MEXICO        SAN LUIS POTOSI
     1937 JUNE 20
          724487 MANIHOT ESCULENTA       MEXICO        GUERRERO
     1937 NOV. 19
          725800 MANIHOT ESCULENTA       MEXICO        GUERRERO
```

```
MATUDA E.
     1934 JULY 9
          650063 MANIHOT PARVICOCCA     MEXICO        CHIAPAS
     1935 SEPT. 7
          688105 MANIHOT ESCULENTA      MEXICO        SAN LUIS POTOSI
```

In another situation the museum worker may wish to look at some specimens, but he is not sure of how to specify the subset of interest, so he poses a broad query where he may expect say 800 instead of the 8 items in our last example. He can then use the printout to help him narrow further the subset, pose another query, if necessary, and continue in this way taking better and better approximations until he has few enough items selected that he can turn conveniently to the specimens themselves.

Sometimes it is desirable to print a reference book of an entire data bank. In museum applications, for example, it is possible to print the catalog in this way. In fact several books can be printed, each ordered on a different leading descriptor, such as a book ordered on catalog number, a book ordered on country of collection, a book ordered on taxonomic name, etc. These can be reprinted periodically and distributed to interested cooperating museums and libraries, thus making up-to-date catalogs widely available to other workers in the field.

If the descriptor list for such a book is complicated, it is a good idea to print first a small sample of the book by using a boolean expression that is guaranteed to specify no more than about a dozen items. The descriptor list with its formating options can then be varied and another small sample printed, until a satisfactory, easily readable printout is obtained. Then using this successful descriptor list and a boolean expression of the type P OR NOT P (e.g., MONTH, JAN. OR NOT MONTH, JAN.) the entire book can be printed.

STATEMENT TYPE: GENERATE

GENERATE noise: descriptor list bFORb noise $\left\{\begin{matrix}\text{bWITHb}\\ \text{bHAVEb}\end{matrix}\right\}$ boolean exp.*

This statement is very much like statement type PRINT, except that instead of a printout, a machine-readable file is generated.

(This file will be on tape or disk or some other external device, depending on the options available at the user's computer installation.) This file has been designed to serve as input to a package of statistical analysis routines.

The same hierarchical ordering rules apply to the descriptor list as in statement type PRINT, but the various printout options (line combos and print field overrides) are ignored. The SAME option applies and refers to the descriptor list of the previous GENERATE or PRINT statement.

Inasmuch as the purpose of this file is to undergo statistical analysis, only numeric descriptor-states are generated. Descriptors known to be numeric (i.e., defined under the FROM-TO option) will have their states transmitted to the file as floating point values. Descriptors defined under the ORDER option with an alphanumeric list of descriptor-states will have the code numbers of their states (not their alphanumeric names) transmitted to the file as floating point values. Descriptors defined under the NAME option are not permitted in the descriptor list.

File Structure

The first record in the file consists of a single integer whose value, n, = the number of descriptors in the descriptor list = the number of words per record in the subsequent records of the file.

Following this first record there is a record (n words long) for each item in the set specified by the boolean expression. These n words are the floating point values of the states of each descriptor in the descriptor list (in descriptor list order) for a single item.

Following these is an end-of-file mark.

Each GENERATE statement creates one such file, and in any one Taxir run, these files are stacked up on the same output device (i.e., tape, disk, etc.). After the last file an extra end-of-file mark is written.

Example: Here is part of a DEFINE DESCRIPTORS statement for a bank of weather information: MONTH (ORDER, JAN., FEB., MAR., APR., MAY, JUNE, JULY, AUG., SEPT., OCT., NOV., DEC.), DAY (FROM 1 TO 31), MAXIMUM TEMPERATURE (FROM -30 TO 120 IN DEG. F.), MINIMUM TEMPERATURE (FROM -50 TO 100 IN DEG. F.). And here is a query to this bank.

PRINT: MONTH, (DAY, MAXIMUM TEMPERATURE, MINIMUM TEMPERATURE) FOR ITEMS WITH WEATHER STATION, 1 AND YEAR, 1965 AND MONTH, JUNE OR JULY AND MAXIMUM TEMPERATURE, FROM 80 TO 120*

We might get the following printout:

```
JUNE
      5  86 DEG. F.   62 DEG. F.
      7  82 DEG. F.   65 DEG. F.
     12  85 DEG. F.   61 DEG. F.
JULY
      8  82 DEG. F.   68 DEG. F.
     24  88 DEG. F.   65 DEG. F.
```

The equivalent GENERATE statement, i.e., the same statement with the word GENERATE substituted for the word PRINT, will yield a file with the following structure and content:

```
Record 1 = 4
Record 2 = 6.0  5.0  86.0  62.0
Record 3 = 6.0  7.0  82.0  65.0
Record 4 = 6.0 12.0  85.0  61.0
Record 5 = 7.0  8.0  82.0  68.0
Record 6 = 7.0 24.0  88.0  65.0
End-of-file mark
```

Other Possible Applications

At the CDC 6400 installation at the University of Colorado a package of statistical analysis programs has been adapted to accept these Taxir generated files as input. Coupling Taxir to the statistics package provides users at this installation with a great flexibility in choosing data samples for analysis.

In a similar manner other data processing packages can be linked to Taxir. We can envision Taxir serving as the central hub of an information management system with Taxir handling the basic maintenance of the data bank (i.e., input of raw data and updating) as well as the selection of data subsets either for direct printout or for further computation by the various ancillary systems attached. The particular file structure described above was designed with the statistics package in mind, but it is not difficult to vary the output structure to interface with other subsystems, including those which accept data in alphanumeric form.

---o---

4. CORRECTING A TAXIR DATA BANK

STATEMENT TYPE: CORRECTION

With this statement the user may correct errors or update obsolete information in his data bank. The general format is:

CORRECTION (d,ds) (d,ds)...(d,ds) noise $\left\{\begin{matrix}\text{bWITHb}\\ \text{bHAVEb}\end{matrix}\right\}$ boolean expression*

d is any descriptor name chosen from the control vocabulary and the ds paired with it is the name of one of the states of that descriptor. If d has been defined under the ORDER or FROM-TO options, then ds must be chosen from the control vocabulary or be the state UNKNOWN. If d has been defined under the NAME option, then ds may be chosen from the control vocabulary or be the state UNKNOWN or be a new name which will be added to the control vocabulary during the correction procedure.

Every item in the set specified by the boolean expression is reassigned to the descriptor-state ds for the descriptor d with which it is paired. This is done for all d,ds pairs included in the statement, which may number from one to as many pairs as there are descriptors defined for the data bank.

Examples:

CORRECTION (GENUS, VOLVOX) FOR ALL ITEMS WITH GENUS, VOLVEX*

CORRECTION (INSTRUCTOR, ASQUITH) (BUILDING, CHEMISTRY) (ROOM, 302)
FOR COURSES WITH DEPARTMENT, CHEMISTRY AND COURSE NO., 400 OR 401*

CORRECTION (LOCATION, UNKNOWN) FOR ITEMS WITH CATALOG NO., FROM
104622 TO 104629*

In general it is good to define one's descriptors so that any single item in the bank may be selected by a boolean expression. This not only permits great flexibility in querying, but also permits corrections of individual items, such as:

CORRECTION (MINIMUM TEMPERATURE, -12) FOR THE ITEM WITH WEATHER
STATION, 5 AND YEAR, 1965 AND MONTH, JAN. AND DAY, 15*

CORRECTION (SIZE, 18) (COLOR, GREEN) (STYLE, K39) FOR THE ITEM
WITH ITEM NO., 173354*

STATEMENT TYPE: DELETE STATE

DELETE STATE d,ds*

The descriptor-state ds of the descriptor d is removed from the control vocabulary, if 1) descriptor d has been defined under the NAME option, and 2) there are no items in the bank assigned to state ds, and 3) ds ≠ UNKNOWN.

Example: Typically this statement type is used to clean spurious states out of the control vocabulary. After a correction like: CORRECTION (GENUS, VOLVOX) FOR ALL ITEMS WITH GENUS, VOLVEX* the misspelling can be purged from the control vocabulary by: DELETE STATE GENUS, VOLVEX*

STATEMENT TYPE: DELETE ITEMS

DELETE ITEMS noise $\left\{\begin{matrix}\text{bWITHb}\\ \text{bHAVEb}\end{matrix}\right\}$ boolean expression*

Every item in the set specified by the boolean expression is removed from the data bank. This very powerful statement should be used with caution. It is possible, easy in fact, to wipe out an entire data bank with one statement. The only way to restore deleted items is to define them again from scratch (unless a reserve copy of the data bank is maintained).

Despite the danger this statement type is quite useful, especially in data banks with a large turnover of information.

Example: In a warehouse inventory bank, new items are continually being defined or deleted as shipments arrive at or leave the warehouse. This procedure must be kept faithfully up to date if there is to be any hope of getting accurate replies to queries asking how many gidgets are currently in stock. Every time a shipment is made, a statement such as one of the following can be addressed to the bank:

DELETE ITEMS WITH SHIPPING NO., 47298*

DELETE ITEMS WHICH HAVE CUSTOMER, GOLDBERG AND ARTICLE, GIDGET OR WIDGET*

DELETE ITEMS WITH YEAR OF SHIPMENT, 1970 AND MONTH OF SHIPMENT, AUG. AND DAY OF SHIPMENT, 7*

The Taxir system responds to statements of this type with 3 lines of printout giving the former item count, the number of items deleted and the current item count.

Example:

```
FORMER NO. OF ITEMS IN THE DATA BANK =  47312
NO. OF ITEMS DELETED =                     82
CURRENT NO. OF ITEMS IN THE DATA BANK = 47230
```

STATEMENT TYPE: DEFINE MORE DESCRIPTORS

DEFINE MORE DESCRIPTORS d(p), d(p),...d(p)*

This statement type permits the user to add new descriptors to a data bank already in existence. It is exactly like DEFINE DESCRIPTORS in all particulars except the following:

1. The addition of the word MORE.
2. Although a DEFINE DESCRIPTORS statement appears only once in the life of a data bank, DEFINE MORE DESCRIPTORS statements may appear as often as desired.
3. Descriptor numbers continue from that of the last descriptor already defined. For example, if in some data bank with 10 descriptors a DEFINE MORE DESCRIPTORS statement adds 3 more descriptors, then these additions will be descriptors 11, 12 and 13.
4. Any items defined after the appearance of a DEFINE MORE DESCRIPTORS statement must include states for the new as well as the old descriptors. Any items defined previous to such a statement will automatically be assigned to the UNKNOWN states of the new descriptors. These can be updated by means of CORRECTION statements.
5. When using the equals facility it is permissible to refer to descriptors defined in the original DEFINE DESCRIPTORS statement or in previous DEFINE MORE DESCRIPTORS statements.

---o---

5. MISCELLANEOUS TAXIR STATEMENTS

STATEMENT TYPE: ID

```
ID noise
```

noise = any string of characters.

Every Taxir program must begin with an ID statement. This statement appears as a header at the top of every page of Taxir output and serves as a means of identifying the printed output with the user. Typically the user will include in his ID statement his name or a brief description of the job. The date need not be included as this is added to the header line by the system, as is a page number.

ID statements may be inserted as often as desired in the program. Each time an ID statement is encountered in a Taxir program, the system adopts it as the header line and begins a new page, repaginating from 1.

Examples:

```
ID---PONSONBY, QUERIES OF DATA BANK Q4
ID: ADDING SOME ITEMS TO CLIMATE BANK, J. CLARK
```

Sometimes it is desirable to physically separate portions of the printed output, as for example, when a number of queries are run together but the printout from each query is to be sent to a different person. To guarantee that such separations can always be made at the perforation between printout sheets, place an ID statement just before and just after any section of the program to be so isolated.

STATEMENT TYPE: END

```
END
```

This statement informs the system that the program is finished. It must appear once and only once in each Taxir program as the last statement of the program. It also prints the total time for the run.

STATEMENT TYPE: TIME

```
TIME noise
```

noise = any string of characters.

This statement type causes a printout of the elapsed time in seconds and milliseconds since the last TIME statement (or if this is the first such statement in the program, since the beginning of the run). Both central processor and peripheral processor times are given.

Examples:

```
TIME
TIME TO ANSWER THE ABOVE 6 QUERIES
TIME TO BUILD BRYOPHYTE DATA BANK
```

STATEMENT TYPE: MEMO

```
MEMO noise*
```

noise = any string of characters except asterisk (*).

This statement type provides the user with a means of embellishing his output with commentary. MEMO statements may be inserted anywhere in the program. The system merely copies them into the output.

Examples:

```
MEMO: THE FOLLOWING 3 QUERIES ARE FOR THE ATTENTION OF MR. ARBUTHNOT*
MEMO TO LYDIA, I ADORE YOU, REGINALD*
                    MEMO
                   LYDIA
                I ADORE YOU
                  REGINALD
                     **
```

STATEMENT TYPES: READ DATA BANK and WRITE DATA BANK

READ DATA BANK
WRITE DATA BANK

Data banks are permanently stored on tape, disk or some other external device, depending on the options available at the user's computer installation. It is necessary to request (in the control language of the operating system in use at the user's installation) that the external device in question be connected to the computer.

READ DATA BANK causes Taxir to read the data bank from the external device into the computer, where it is then available for Taxir processing.

WRITE DATA BANK causes Taxir to write the data bank from the computer onto the external device, thus preserving it for future use.

If any of the following data bank altering statements appear in a Taxir program, a WRITE DATA BANK statement should appear after the last of them in order to capture these efforts for later use.

DEFINE DESCRIPTORS
DEFINE MORE DESCRIPTORS
DEFINE ITEMS
DEFINE AND PRINT ITEMS
CORRECTION
DELETE STATE
DELETE ITEMS

If none of the above statements appear, a WRITE DATA BANK statement is not necessary, as the bank still exists in its current form on the external device.

In any run in which a bank is built for the first time (i.e., contains a DEFINE DESCRIPTORS statement) no READ DATA BANK statement is necessary or useful. In all other runs a READ DATA BANK statement is needed prior to any processing of the bank. Only one Taxir data bank can be handled in any one Taxir run.

Note that WRITE DATA BANK wipes out the data bank stored on the external device by replacing it with the current bank in the computer. Only one WRITE DATA BANK statement is needed in any one program. If more than one such statement appears, only the bank at the time of the last such statement will be preserved, as each WRITE DATA BANK statement will wipe out the effect of the previous one.

There is a convenience in this feature. If the Taxir language is properly used, the external device will always contain the most up-to-date bank at the end of each run. There is also a danger. If the statements which modify the bank are disastrously invalid (such as an accidental DELETE ITEMS statement that wipes out the bank), then the new-replaces-old feature can cause considerable hardship. As a safety procedure, it is good to maintain a reserve copy of the data bank. This can be done through the control language of the operating system following each run which updates the bank (i.e., contains a WRITE DATA BANK statement).

---o---

APPENDIX A: TAXIR ERROR MESSAGES

During the processing of Taxir statements the system checks for a large number of user errors. When such errors are encountered Taxir prints out a message of the type:

ERROR TYPE #

If the user receives such a message from Taxir he should look up the error # in the following list to discover the nature of the difficulty.

Any error arising from a DEFINE DESCRIPTORS, DEFINE MORE DESCRIPTORS, DEFINE ITEMS, DEFINE AND PRINT ITEMS or READ DATA BANK statement causes the run to terminate. There is no point in continuing the run if there is a faulty or no data bank in the machine.

All other errors cause the system to skip over the balance of the faulty statement and continue with the next Taxir statement. In the process of skipping over the faulty statement any other errors in the same statement will be overlooked, so it is a good idea not only to correct the error that is caught by the system, but to check the entire statement for validity before resubmitting the Taxir program.

When an error is caught in an Item Definition, the entire item is rejected by the system and does not get into the data bank. Likewise, when an error is caught in a CORRECTION statement, none of the corrections requested is made. However, any new state names that are introduced in a faulty Item Definition or CORRECTION statement, before the point at which the error is detected, will get into the control vocabulary. If such names are not valid, they may be removed from the control vocabulary by means of DELETE STATE statements. Any Item Definition or CORRECTION statement rejected on errors may, of course, be corrected and resubmitted to the system.

Taxir Error List

1. Unrecognizable statement name.

2. Statement name not complete on first card of statement.

3. FROM-TO descriptor with a non-positive BY parameter.

4. Too many descriptors in a PRINT or GENERATE descriptor list or too many (d,ds) pairs in a CORRECTION statement. Reduce their number or recompile Taxir, enlarging tables DLIST (1st dimension), ITEM (1st dimension) and STAT, as well as the constant DLXMAX. Set their values = each other and ≥ the number of descriptors in the data bank.

5. Descriptor name must not begin with a left parenthesis.

6. Too many descriptors being defined. Reduce their number or recompile Taxir, enlarging tables DD, DD1, and DD2, as well as the constant DDMAX. Set their values = each other and ≥ the number of descriptors desired.

7. Too many descriptor or descriptor-state names containing > 10 characters. Reduce their number or recompile Taxir, enlarging table OVRFLO and constant OVMAX. Set their values = each other and ≥ the expected number of descriptor and descriptor-state names that contain between 11 and 20 characters + twice the expected no. of names between 21 and 30 characters + 3 times the expected no. of names between 31 and 40 characters + etc.

8. Not enough space dimensioned in the Taxir system for descriptor-states. Either reduce the estimates on descriptors under the NAME option or recompile Taxir, enlarging tables DSS, DSCODE and the constant DSSMAX. Set their values = each other and ≥ the sum of the estimates on descriptors under the NAME option + the sum of the number of states in descriptors defined under the ORDER option.

9. Coded descriptor-state out of the range defined.

10. Equals reference to prior descriptor not valid.

11. Neither NAME, FROM-TO nor ORDER option is specified.

12. No number found where expected.

13. 90 character limit on name length exceeded.

14. All blank name is not permitted.

15. Trying to define the same descriptor name twice.

16. Using a descriptor name never defined to Taxir.

17. Trying to define the same descriptor-state name twice.

18. Using a descriptor-state name never defined to Taxir.

19. No right parenthesis following descriptor parameters.

20. Not enough space reserved for base characteristic functions in the data matrix. Reduce the size of some of the descriptors or recompile Taxir, enlarging table IFILES (1st dimension) and the constant DMAX.

21. Too many printout lines per item specified in a descriptor list. Reduce their number by combining descriptors into line combos or recompile Taxir, enlarging table COMBO and the constant COMMAX.

22. No closing parenthesis on a line combo in a descriptor list.

23. No comma or asterisk after coded descriptor-state.

24. No. of states in this descriptor has exceeded the estimate given in the DEFINE DESCRIPTORS statement. At the present time there is no remedy for this other than to redefine that data bank from scratch.

25. This is not a user error, but is an indicator of trouble in the Taxir system. Save all printout and input cards and contact the author.

26. Like #18.

27. FROM-TO descriptor with decreasing FROM-TO range.

28. Estimate of number of states in a descriptor defined under NAME option is not a positive integer.

29. Statement name not followed by asterisk or FOR.

30. Boolean expression too long for processing space reserved. Break up query into a series of smaller queries or recompile Taxir, enlarging table STRYNG and constant STRGMX.

31. Boolean expression too long for processing space reserved. Break up query into a series of smaller queries or recompile Taxir, enlarging table STACK and constant STKMAX.

32. Boolean expression does not have balanced parentheses, i.e., does not have same number of left and right parentheses.

33. Trying to operate on a data bank that is not present in the machine, i.e., which has either not been defined earlier in the run or has not been read into the machine from an external device.

34. Cannot delete a state from a descriptor whose option is other than NAME.

35. A descriptor with option NAME has been included in the descriptor list of a GENERATE statement.

36. Taxir is not properly dimensioned to hold this data bank. One or more of the critical values printed for the stored data bank exceeds the maximum permitted by the Taxir in use. Use the Taxir which generated the stored bank or redimension the one in use to accomodate the excess.

37. Neither CARDS nor TAPE nor DISK specified.

38. Cannot delete the UNKNOWN state from a descriptor.

39. Illegal FROM-TO range in a boolean expression.

40. FROM value >TO value in a boolean expression.

41. Boolean expression must not begin with AND or OR.

42. Illegal use of parentheses in a boolean expression.

43. TO missing from FROM-TO range in boolean expression.

44. Illegal operator in a boolean expression.

45. NOT or FROM expected but not found in a boolean expression.

46. NOT expected but not found in a boolean expression.

47. Like #25.

48. Like #46.

49. Like #42.

50. Expecting a FROM-TO descriptor-state, it turns out to be neither numeric nor UNKNOWN.

51. FROM-TO descriptor-state not in the defined set of numbers.

52. In a descriptor list the left parenthesis denoting the beginning of a line combo has occurred before the right parenthesis denoting the end of the previous line combo.

53. Right parenthesis missing or TO expected but not found in a boolean expression.

54. Illegal termination of a boolean expression.

55. AND, OR, NOT or TO expected but not found in a boolean expression.

56. Like #42.

57. Operator expected but not found in a boolean expression.

58. AND or OR expected but not found in a boolean expression.

59. The right parenthesis denoting the end of a line combo has occurred with no previous left parenthesis.

60. Like #25.

61. No left parenthesis before (d,ds) pair.

62. No descriptor name or descriptor-state name where expected.

63. Asterisk encountered too soon or left parenthesis out of place.

64. A new descriptor-state has been introduced in a descriptor whose option is ORDER or FROM-TO.

65. Too many FROM-TO descriptors. Reduce their number or recompile Taxir, enlarging tables FROM, TO, BY, IN and constant FTBIMX. Set their values = each other and ≥ the number of FROM-TO descriptors desired.

66. FROM-TO descriptor without a TO parameter.

---o---

APPENDIX B: PRELIMINARY PROBLEMS

Data Bank Design

In Chapter 1 there is a list of six questions which the potential Taxir user must ask and answer before he can create Taxir data banks. In some simple cases answering these questions will be obvious and the answers will be jotted down on a single sheet of paper. In many cases, however, these will prove to be difficult decisions complicated by cost factors, manpower availability, the nature of the computer installation (its equipment, operating system and policies) and by other special factors known only to the potential user.

Here then are some additional questions to be considered:

What will be the nature of information flow and work flow in my shop? This includes the gathering and transcribing of data, the setting up of computer runs for building, updating and querying data banks, and the distribution and application of computer results. The answer to this question in any complex situation will probably take the form of flow diagrams.

What are my financial capabilities and limitations? And what will my operations cost? The answer to these questions may take the form of a cost analysis and/or a cash flow analysis, and possibly after the operation has begun, a cost-effectiveness analysis.

Such analyses invariably raise a host of other related questions which may influence the final data bank design. For example, the user may wish to include a certain descriptor but may discover that the cost of gathering this information outweighs the expected benefits. Or on the contrary, he may decide that in view of his goals this descriptor is essential at all costs, but because of its high cost, some other less vital descriptor(s) must be sacrificed.

The design headaches can in some cases be considerable, especially if one's goals exceed one's resources. The six questions included in the text are meant to help the user define his goals. The questions that are raised here in this appendix are meant to help the user define his resources. The final answers to the goal questions will often be tempered by the answers to the resource questions, and for this reason all of these questions must be considered simultaneously and in the total context of the problem and its environment. In any complex data management situation the Taxir system (or any other system for that matter) is only one link in a larger system of information, money and manpower flow. The potential Taxir user who needs help is advised to call in a systems analyst with Taxir experience.

Fitting the Taxir System to the Data Bank

The Taxir system is quite general in its powers, however, the system must always operate within the storage space limits of some particular configuration of computer hardware. The Taxir system itself occupies some of this space and the rest is used by Taxir as working space, i.e., for holding various data parameters, data elements, dictionaries, etc. The amount of space needed for each of these tasks will vary from bank to bank according to a set of formulas involving the number of items, the number of descriptors, the number of states in each descriptor, the options chosen, etc. In order to allocate this space to best advantage it is the policy of the Taxir designers to fit the Taxir system on a custom basis to each data bank in the context of a specific computer configuration.

The Taxir system is written in the Fortran programming language and the working space in question takes the form of various Fortran arrays. Fitting the system to the data bank consists of setting the dimensions of these arrays large enough to accomodate the data and small enough to fit within the hardware limitations. If the space limits are exceeded the user may decide to choose larger hardware, to decrease the size of the data bank or to store some of these arrays on auxiliary storage devices and process them a piece at a time. All of these decisions will affect the cost of operation which may in turn force modifications in the data bank design.

In any case, to fit the system to the data bank advantageously a good understanding of Taxir space allocation is required. Here again a systems analyst with Taxir experience can be of great help. This primer treats only the general powers of Taxir (those which are invariant on all data banks and on all machines) and the problems involved in data bank fitting will vary from bank to bank and from installation to installation. A Taxir analyst can either perform this task for the potential user or prepare for him a document showing him how to do this for himself. If the user's estimates of his data dimensions are not too small, this job need be done only once per bank, just prior to the actual creation of the data bank. As may be seen from a perusal of the Taxir Error List (Appendix A), if any of the array sizes are exceeded during the course of operations, an error message is issued by the system indicating which array(s) must be enlarged.

Running Taxir under the Operating System

In the present era of electronic computing almost all computer installations prohibit the direct hands-on use of the equipment by anyone but professional machine operators. The computer's activities are under the combined control of the operators and a master program called the operating system which coordinates the runs of the various users and provides them with a variety of auxiliary computing services.

Under the most sophisticated operating systems two modes of operation are permitted, time-share mode and batch mode.

In time-share mode a large number of users (up to about 100 nowadays) at remote terminals are able to use the computer simultaneously and independently. From his own point of view such a user appears to have the entire machine to himself. In particular, if the program he is using is so designed he can engage in a two way communication with his program, often at human reaction time speeds. Actually the operating system is continually rescheduling the use of the machine and running little snips of various programs so as to maximize the overlapping of operations. While user A is pondering what to do next, user B may be computing, user C may be getting a printout over his terminal and user D may be typing a message to his program, while the operating system runs the computer like an overworked short order cook trying to keep all his customers happy at once.

This computing environment is ideal for the Taxir system. Since each Taxir statement is executed independently of any other Taxir statement, the user may freely dialogue with Taxir. For example, he may type a query on his terminal keyboard and see the response before typing the next query which may be a modification of the first query. In this context the RESULT operand (p. 27) becomes very powerful for rapidly taking better and better approximations of a desired subset.

In batch mode the operating system does some overlapping of operations, but no rapid dialogue is possible. The user submits his entire run at once (usually in the form of a card deck) and patiently waits for the entire output of the run. Dialogue is possible only in slow motion by making a series of runs. Batch mode is ideal for runs with no need for man-machine interaction, for example, a Taxir run for entering a large series of Item Definitions into a data bank.

There are numerous advantages both to an installation and to its users in running under an operating system, whether in batch or time-share modes. The principal price that the user must pay for this service is that he must learn (at least a part of) the control language of the operating system so that he can express his wishes and requirements to the system and at the same time conform to the demands the system places on him. With respect to Taxir the user's wishes will consist largely of requests to hook up external devices (tapes, disks, etc.) where the Taxir system and the user's data banks permanently reside when not in use. The demands of the operating system on the user will vary according to the operating system, but as a minimum, the user is usually required to submit an estimate of the time and space requirements of his run.

All of this is to inform the naive potential Taxir user that the operating system is a fact of life that must be dealt with. This is not especially difficult but it is necessary. The user will have to familiarize himself with the basics of his operating system or have available the services of someone who has already done so. Here again the Taxir systems analyst can help, but he must also be well versed in the operating system of the user's particular installation.

Taxir System Conversion

The Taxir system is written in Fortran and although Fortran programs run on virtually every computer in the world, intrinsic hardware design differences among different computers have rendered it impossible for programs written for one machine to run on another machine without some reprogramming, despite the universality of the Fortran language. In some cases this reprogramming is minor. In other cases a considerable conversion effort is required. In still other cases the effort is not worthwhile as the efficiency of the program is disastrously degraded in the process.

The potential Taxir user may avoid this problem altogether if he chooses a computer on which Taxir is already operative. However, sometimes a user's choice of equipment is limited severely by what his circumstances may dictate. If the user must operate on a computer for which there as yet exists no Taxir system, then he must consider first the technical feasibility of a conversion and secondly its cost. Only a skilled programmer should attempt such a conversion.

At the time of this writing, the only complete version of Taxir operates on the CDC 6400 in batch mode only. A limited (very early) version operates on the IBM 360/65 with a choice of time-share or batch modes. Plans have been made and are awaiting funding for a full version to run on the IBM 360/67 with a choice of time-share or batch modes. Plans also exist for adding time-share capability to the CDC 6400 version, now that a time-share operating system is available for that computer.

Summary

The main point of this appendix is that getting started is the most difficult part of using Taxir. Data design, cost analysis, hardware limitations, fitting the system to the data bank, learning to use the operating system, and sometimes Taxir system conversion - these problems must be solved before Taxir can operate. It is in these areas that the potential user, especially if he is inexperienced, can most benefit from the help of the professional. However, it is the author's firm belief that once the intelligent but inexperienced user has been helped over these preliminary hurdles, he can easily use Taxir to manipulate his data without further help, solely by drawing on this primer and his subsequent experience.

Please refer any problems to the author, R. C. Brill, 1304 8th St., Boulder, Colorado 80302. Phone: 303-443-6160. I will be glad to help you or recommend some other qualified person to help you.

---o---

INDEX

A

a (as in FROM a TO b) 12
alphabetic ordering iv, 34
alphanumeric mode 14
AND (intersection) 9, 21-23
asterisk (*) 6, 8, 21, 56

B

b (as in FROM a TO b) 12
b (required blank) 7
batch mode 66
BCD mode 14
blanks
    embedded 6, 9
    leading 6, 9
    required 6, 7
    trailing 6, 9
boolean algebra 2
boolean expressions 22, 27
boolean operands
    type 1 (d,ds) 24
    type 2 (d,ds OR ds OR... OR ds) 25
    type 3 (d, FROM ds1 TO ds2) 25
    type 4 (RESULT) 9, 21, 27, 66
boolean operators
    complement (NOT) 23
    intersection (AND) 23
    union (OR) 23
brackets (in general formats) 7
building data banks 8, 18
BY 12

C

c (as in Fixed Field Format) 14, 15
c (as in FROM a TO b BY c) 12
CARDS 14
cards 5, 6
CDC 6400 51, 68
character 8
code numbers 10
colon (:) 33
combining operators and operands into boolean expressions 27
comma (,) 8, 21
complement (NOT) 23
contains (as in set D contains set E) 3
control language of operating system 14, 57, 58, 67
CONTROL VOCABULARY 21
correcting data banks 52
CORRECTION 22, 52, 57, 59
cost analysis 64

D

d (descriptor name) 8, 9, 11-13, 21, 24-26, 52-54
dashes (---) 34, 41
data bank (defined) 3
data bank design considerations 5, 64
date 55
decimal point (.) 12, 18
DEFINE AND PRINT ITEMS 17, 57, 59
DEFINE DESCRIPTORS 8, 22, 57, 59
DEFINE ITEMS 8, 14, 57, 59
DEFINE MORE DESCRIPTORS 22, 54 57, 59
DELETE ITEMS 53, 57
DELETE STATE 22, 53, 57, 59
descriptor (defined) 4
descriptor lists 33
descriptor names 8, 13
descriptor number 13, 54
descriptor-state (defined) 4
descriptor-state names 9
disjoint (sets) 3
DISK 14
disk 57
ds (descriptor-state name) 9-11, 17-18, 24-25, 27, 52, 53
ds1 10, 11, 25-27
ds2 10, 11, 25-27
d,ds (operand type 1) 24
d,ds OR ds OR...OR ds (operand type 2) 25
d, FROM ds1 TO ds2 (operand type 3) 25

E

element (of a set) 1
ellipsis (...) 7
embedded blanks 6, 9
empty set (Ø) 1
END 55
END OF ITEMS 8, 14, 18
end-of-file mark 14, 50
equal sign (=) 13
equals facility 13, 54
error list 60
error messages 59
essential parentheses 31
est (estimate of no. of states) 11
establishing membership of a set 3
external device 57, 67
    cards 5, 6, 14
    disk 14, 57
    tape 14, 57
    terminal 5

F

f (as in Free Field Format)
    14, 16
F parameter (follow) 43
field 15-17
field order no. 16
fitting Taxir to data bank 65
FIXED 14
Fixed Field Format 15
FOR 9, 21, 33, 49
FREE 14
Free Field Format 16
FROM 9, 12, 21, 25
FROM-TO Option 12, 18

G

general formats 7
GENERATE 49

H

HAVE 32, 33, 49, 52, 53
header line 55
hierarchy of descriptors in
    query response iv, 34
HOW MANY 32

I

i (descriptor no.) 13
IBM 360 68
ID 55
IN 12
indentation of query
    response 39
information flow 64
intersection (AND) 23
item (defined) 3
Item Definition 8, 14, 15,
    17, 22, 59

J

j (descriptor name) 13
justification
    left 40-43
    right 40-43

L

L parameter (left-justify) 42
label 12, 25
language, Taxir iv, 2, 5
leading blanks 6, 9
left justification in query
    response 40-43
left parenthesis [(] 8, 21
line combos 37

M

maximum print field length 41
member (of a set) 1
MEMO 56
minimum print field length 41
minus sign (-) 12, 18
mutually exclusive sets 4

N

NAME Option 11, 18
noise 14, 32, 33, 55, 56
noise terminator 14, 32, 33
NOT (complement) 9, 21-23
null set (Ø) 1
number of cards per statement 6
numeric ordering iv, 34

O

omission of duplicate states in
    query response 37, 38
operands (see boolean operands)
operating system 66
operators (see boolean operators)
options (see F parameter, Fixed
    Field Format, Free Field Format,
    FROM-TO Option, L parameter,
    NAME Option, ORDER Option,
    R parameter, SAME Option)
OR (union) 9, 21-23
ORDER Option 9, 18
ordering of descriptor-states in
    query response 33
overlap (of sets) 3

P

p (parameter list) 9
page number 55
parentheses
    essential 31
    in boolean expressions 28
    in descriptor lists 37, 42
    left 8, 21
    right 8, 21
pictorial representation of sets 2
power set 22
PRINT 33
print field override options
    F parameter (follow) 43
    L parameter (left-justify) 42
    R parameter (right-justify) 42
print fields 40
priority of operations in boolean
    expressions 30
programs, Taxir 5

Q

- query response (see rules)
- querying data banks 21, 45

R

- R parameter (right-justify) 42
- READ DATA BANK 57, 59
- required blanks 6, 7
- reserve copy of data bank 53, 58
- RESULT (operand type 4) 9, 21, 27, 66
- right-justification in query response 40-43
- right parenthesis [)] 8, 21
- rules for
    - ...combining operators and operands into boolean expressions 27
    - ...descriptor names 8
    - ...descriptor-state names 9
    - ...establishing membership of a set 3
    - ...labels 12
    - ...parentheses in boolean expressions 28
    - ...printout of query response
        - hierarchy of descriptors 34
        - indentation 39
        - left-justification 40-43
        - maximum print field length 41
        - minimum print field length 41
        - omission of duplicate states 37, 38
        - ordering of descriptor states 33
        - right-justification 40-43
        - truncation 39
    - ...priority of operations in boolean expressions 30
    - ...statements 6

S

- SAME 9, 14
- SAME Format 17
- SAME Option 44
- set (defined) 1
    - contains 3
    - disjoint 3
    - element 1
    - empty 1
    - establishing membership 3
    - member 1
    - mutually exclusive 4
    - null 1
    - overlap 3
    - pictorial representation 2
    - power set 22
    - subset 1, 3
    - theory iv, 1
    - universal 1
- state (defined) 4
- statement names 5-7
- statement types 5
- statistical analysis 50
- subset 1, 3
- system conversion 67

T

- TAPE 14
- tape 57
- Taxir
    - boolean operands 24
    - data banks 3
    - error list 60
    - error messages 59
    - general description iv
    - history v
    - language iv, 2, 5
    - operands 24
    - programs 5
    - statements 5
    - system conversion 67
    - user 5
- terminal 5
- three dashes (---) 34, 41
- TIME 56
- time-share mode 66
- TO 9, 12, 21, 25
- trailing blanks 6, 9
- truncation of query response 39

U

- U (universal set) 1
- union (OR) 23
- universal set (U) 1
- UNKNOWN state 18, 21, 24, 52-54
- updating data banks 52

W

WITH 32, 33, 49, 52, 53
work flow 64
WRITE DATA BANK 57

Special Symbols

* (asterisk) 6, 8, 21, 56
{} (brackets) 7
: (colon) 33
, (comma) 8, 21
. (decimal point) 12, 18
... (ellipsis) 7
= (equal sign) 13
((left parenthesis) 8, 21
- (minus sign) 12, 18
Ø (null set) 1
) (right parenthesis) 8, 21
--- (three dashes) 34, 41